

AD-A241 692



ANNUAL REPORT

VOLUME 1

PART 2

TASK 1: DIGITAL EMULATION TECHNOLOGY LABORATORY

REPORT NO. AR-0142-91-001

September 27, 1991

---

DIGITAL EMULATION TECHNOLOGY LABORATORY

Contract No. DASG60-89-C-0142

Sponsored By

The United States Army Strategic Defense Command

---

COMPUTER ENGINEERING RESEARCH LABORATORY

Georgia Institute of Technology

Atlanta, Georgia 30332 - 0540

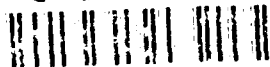
---

Contract Data Requirements List Item A005

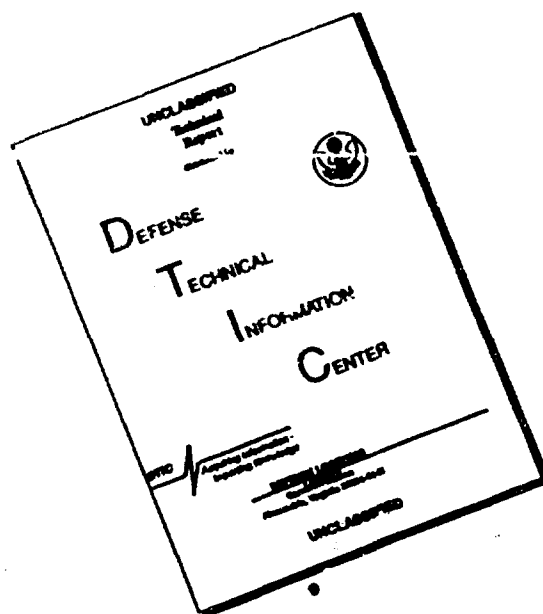
Period Covered: FY 91

Type Report: Annual

91-12579



# DISCLAIMER NOTICE



THIS DOCUMENT IS BEST  
QUALITY AVAILABLE. THE COPY  
FURNISHED TO DTIC CONTAINED  
A SIGNIFICANT NUMBER OF  
PAGES WHICH DO NOT  
REPRODUCE LEGIBLY.

## DISCLAIMER

DISCLAIMER STATEMENT - The views, opinions, and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other official documentation.

## DISTRIBUTION CONTROL

- (1) DISTRIBUTION STATEMENT - Approved for public release; distribution is unlimited.
- (2) This material may be reproduced by or for the U.S. Government pursuant to the copyright license under the clause at DFARS 252.227 - 7013, October 1988.



Accession For	
NTIS	✓
DTIC	
Unpublished	
Justification	
By	
Distribution	
Approved for release	
Date	
Dist	Spec
A-1	

**ANNUAL REPORT**

**VOLUME 1**

**PART 2**

**TASK 1: DIGITAL EMULATION TECHNOLOGY LABORATORY**

September 27, 1991

---

**Authors**

**Thomas R. Collins and Stephen R. Wachtel**

**COMPUTER ENGINEERING RESEARCH LABORATORY**

Georgia Institute of Technology

Atlanta, Georgia 30332 - 0540

---

Eugene L. Sanders

USASDC

Contract Monitor

Cecil O. Alford

Georgia Tech

Project Director

---

Copyright © 1991

Georgia Tech Research Corporation

Centennial Research Building

Atlanta, Georgia 30332



# TABLE OF CONTENTS

## PART 1

<b>1. Introduction .....</b>	<b>1</b>
1.1. Objectives .....	1
1.1.1. GN&C Test and Evaluation -- EXOSIM.....	3
1.1.2. Education and Technical Support .....	4
1.2. Schedules and milestones .....	5
<b>2. Hardware and Facilities.....</b>	<b>9</b>
2.1. Parallel Function Processor (PFP) .....	9
2.1.1. Physical Description.....	10
2.1.2. Intel 310 Host.....	12
2.1.3. Sun 386i Host .....	12
2.2. Seeker Scene Emulator (SSE).....	12
2.3. Other computer systems .....	13
2.4. Secure laboratory .....	14
<b>3. FPP/FPX Development Tools .....</b>	<b>16</b>
3.1. Introduction.....	16
3.2. FPP/FPX object module loader .....	16
3.3. FPP/FPX program downloader.....	17
<b>4. Software Development Tools .....</b>	<b>18</b>
4.1. Introduction.....	18
4.2. Sequential programming tools.....	18
4.2.1. INITIAL program.....	18
4.2.2. DECLARE program .....	21
4.2.3. STRUCTURE program .....	25
4.2.4. CTIMER program .....	27

4.3. Parallel programming tools .....	31
4.3.1. NETWORK program.....	31
4.3.2. USAGE program.....	37
4.3.3. ETIMER program .....	41
4.4. Special purpose tools.....	45
4.4.1. NAMELIST program .....	45
4.4.2. EQUIVALENCE program.....	46
4.4.3. COMMON program .....	49
4.4.4. PROLOG utility .....	51
<b>5. Application Software.....</b>	<b>63</b>
5.1. EXOSIM.....	63
5.1.1. EXOSIM 1.0.....	65
5.1.2. EXOSIM 2.0.....	66
5.1.2.1. SSV19.3.....	70
5.1.2.2. SSV19.5.....	71
5.1.2.3. SSV19.6.....	74
5.1.2.4. SSV19.7 and SSV19.8 .....	76
5.1.2.5. SSV20.8.....	77
5.1.2.6. SSV20.9.....	78
5.1.2.7. SSV20.10.....	80
5.1.2.8. SSV20.11.....	81
5.1.2.9. SSV20.12.....	85
5.1.2.10. SSV20.13.....	88
5.1.2.11. SSV20.14.....	91
5.1.2.12. SSV20.15.....	94
5.1.2.13. SSV20.16.....	98
5.1.2.14. SSV21.16.....	102
5.1.2.15. SSV22.16.....	103
5.1.2.16. SSV22.19.....	108
5.2. LEAP.....	114
<b>6. Appendix A: Environment file format .....</b>	<b>117</b>
<b>7. Appendix B: vield program source.....</b>	<b>119</b>
<b>8. Appendix C: loadfpp program source.....</b>	<b>135</b>

## **PART 2**

<b>9. Appendix D: common program source .....</b>	<b>1</b>
<b>10. Appendix E: ctimer program source .....</b>	<b>43</b>
<b>11. Appendix F: declare program source .....</b>	<b>109</b>
<b>12. Appendix G: equivalence program source .....</b>	<b>169</b>
<b>13. Appendix H: etimer program source .....</b>	<b>213</b>
<b>14. Appendix I: initial program source .....</b>	<b>285</b>
<b>15. Appendix J: namelist program source .....</b>	<b>328</b>
<b>16. Appendix K: network program source .....</b>	<b>343</b>
<b>17. Appendix L: structure program source .....</b>	<b>393</b>
<b>18. Appendix M: usage program source .....</b>	<b>426</b>

## **PART 3**

<b>19. Appendix N: EXOSIM 2.0 (End-to-end) .....</b>	<b>1</b>
--	----------

## 9. Appendix D: common program source

FILE: common/Makefile

```
#
# Copyright 1991
# Georgia Institute of Technology
# Computer Engineering Research Laboratory
# Author: Stephen R. Wachtel
#

default:    common

CC = cc -g
INCLUDE = include
CFLAGS = -IS$(INCLUDE)
LIBRARY = library/library.a

OBJECTS = \
    $(INCLUDE)/grammar.h \
    *grammar.[co] \
    *scanner.[co] \
    yytrace.[co] \
    y.output

PROGRAMS = \
    *common

grammar.c: grammar.y
    yacc -dv grammar.y
    mv y.tab.h $(INCLUDE)/grammar.h
    mv y.tab.c grammar.c

scanner.c: scanner.l
    lex -vt scanner.l | sed 's/getc/yygetc/' >scanner.c

scanner.o: scanner.c
    $(CC) $(CFLAGS) -c scanner.c

grammar.o: grammar.c
    $(CC) $(CFLAGS) -c grammar.c

common: grammar.o scanner.o $(LIBRARY)
    $(CC) -c common grammar.o scanner.o $(LIBRARY)

sgrammar.c: grammar.c yytoken.awk
    awk -f yytoken.awk <grammar.c >sgrammar.c

sgrammar.o: sgrammar.c
    $(CC) $(CFLAGS) -c sgrammar.c

scommon:    sgrammar.o scanner.o $(LIBRARY)
    $(CC) -o scommon sgrammar.o scanner.o $(LIBRARY)

dscanner.c: scanner.c
    cp scanner.c dscanner.c

dscanner.o: dscanner.c $(INCLUDE)/grammar.h
    $(CC) $(CFLAGS) -DDEBUG -c dscanner.c

dcommon:    grammar.o dscanner.o $(LIBRARY)
    $(CC) -o dcommon grammar.o dscanner.o $(LIBRARY)

tgrammar.c: grammar.c
    sed 's/yystack:/& yytrace(yystate);/' <grammar.c >tgrammar.c

tgrammar.o: tgrammar.c
    $(CC) $(CFLAGS) -c tgrammar.c
```

## 2 Annual Report: Digital Emulation Technology Laboratory Volume 1, Part 2

```
tcommon: tgrammar.o scanner.o yytrace.o $(LIBRARY)
$(CC) -o tcommon tgrammar.o scanner.o yytrace.o $(LIBRARY)
```

```
yytrace.c: grammar.c yytrace.awk
awk -f yytrace.awk <y.output >yytrace.c
```

```
yytrace.o: yytrace.c
$(CC) $(CFLAGS) -c yytrace.c
```

```
clean:
cd library; make clean
rm -f $(PROGRAMS) $(OBJECTS)
```

FILE: common/grammar.y

```
/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */
```

```
/*
 * FORTRAN 77
 */
```

```
%token RW_AND
%token RW_ASSIGN
%token RW_BACKSPACE
%token RW_BLOCK_DATA
%token RW_CALL
%token RW_CHARACTER
%token RW_CLOSE
%token RW_COMMON
%token RW_COMPLEX
%token RW_CONTINUE
%token RW_DATA
%token RW_DIMENSION
%token RW_DO
%token RW_DOUBLE_PRECISION
%token RW_ELSE
%token RW_ELSE_IF
%token RW_END
%token RW_END IF
%token RW_ENDFILE
%token RW_ENTRY
%token RW_EQ
%token RW_EQUIVALENCE
%token RW_EQV
%token RW_EXTERNAL
%token RW_FALSE
%token RW_FORMAT
%token RW_FUNCTION
%token RW_GE
%token RW_GO_TO
%token RW_GT
%token RW_IF
%token RW_IMPLICIT
%token RW_INCLUDE
%token RW_INQUIRE
%token RW_INTEGER
%token RW_INTRINSIC
%token RW_LE
%token RW_LOGICAL
%token RW_LT
%token RW_NAMELIST
%token RW_NE
%token RW_NEQV
%token RW_NOT
%token RW_OPEN
%token RW_OR
%token RW_PARAMETER
%token RW_PAUSE
%token RW_PRINT
%token RW_PROGRAM
```

```

%token RW_READ
%token RW_REAL
%token RW_RETURN
%token RW_REWIND
%token RW_SAVE
%token RW_STOP
%token RW_SUBROUTINE
%token RW_THEN
%token RW_TO
%token RW_TRUE
%token RW_WRITE
%token RW_UNDEFIED

%token COMMENT
%token CONCATENATE
%token DOUBLE_PRECISION
%token EXPONENTIATE
%token HOLLERITH
%token IDENTIFIER
%token INTEGER
%token LABEL
%token REAL
%token STRING

%left ','
%nonassoc ':'
%right '='
%left RW_EQV RW_NEQV
%left RW_OR
%left RW_AND
%left RW_NOT
%nonassoc RW_EQ RW_NE RW_LT RW_LE RW_GT RW_GE
%left CONCATENATE
%left '+' '-'
%left '*' '/'
%right EXPONENTIATE
%left SIGN

%{
typedef char *POINTER;
#define YYSTYPE POINTER

#include "list.h"
LIST* clist = 0;

char *block_name = 0;
char *common_name = 0;

extern POINTER duplicate( );
extern POINTER merge( );
extern POINTER list( );
extern POINTER type( );
extern POINTER replicate( );
%}

%%

program:
    optional_statement_list
    ;

optional_statement_list:
    /* NULL */
    |
    statement_list
    ;

statement_list:
    statement
    |
    statement_list statement
    ;

```

```
statement:
    comment_statement
    |
    label unlabeled_statement
    ;
```

```
comment_statement:
    COMMENT
    ;
```

```
label:
    LABEL
    ;
```

```
unlabeled_statement:
    include_statement
    |
    program_statement
    |
    block_data_statement
    |
    function_statement
    |
    subroutine_statement
    |
    entry_statement
    |
    end_statement
    |
    specification_statement
    |
    executable_statement
    |
    format_statement
    ;
```

```
include_statement:
    RW_INCLUDE character_constant
    ;
```

```
program_statement:
    RW_PROGRAM program_identifier
    ;
```

```
program_identifier:
    IDENTIFIER
    {
        $$ = block_name = $1;
    }
    ;
```

```
block_data_statement:
    RW_BLOCK_DATA block_data_identifier
    ;
```

```
block_data_identifier:
    IDENTIFIER
    {
        $$ = block_name = $1;
    }
    ;
```

```
function_statement:
    RW_FUNCTION function_identifier optional_formal_argument_list
    |
    type RW_FUNCTION function_identifier optional_formal_argument_list
    ;
```

```
function_identifier:
    IDENTIFIER
    {
```

```

        $$ = block_name = $1;
    }
;

subroutine_statement:
    RW_SUBROUTINE subroutine_identifier
;
    RW_SUBROUTINE subroutine_identifier optional_formal_argument_list
;

subroutine_identifier:
    IDENTIFIER
    {
        $$ = block_name = $1;
    }
;

entry_statement:
    RW_ENTRY entry_identifier
;
    RW_ENTRY entry_identifier optional_formal_argument_list
;

entry_identifier:
    IDENTIFIER
    {
        $$ = $1;
    }
;

optional_formal_argument_list:
    '(' ')'
    {
        $$ = 0;
    }
;
    '(' formal_argument_list ')'
    {
        $$ = $2;
    }
;

formal_argument_list:
    formal_argument
    {
        $$ = merge( "${s}", $1 );
    }
;
    formal_argument_list ',' formal_argument
    {
        $$ = merge( "${s}${s}", $1, $3 );
    }
;

formal_argument:
    IDENTIFIER
    {
        $$ = $1;
    }
;
    formal_argument_alternate_return
    {
        $$ = $1;
    }
;

formal_argument_alternate_return:
    '*'
    {
        $$ = duplicate( "*" );
    }
;

```



```

end_statement:
  RW_END
  {
    print_list( clist, block_name );
    delete_list( clist );

    clist = 0;
    block_name = 0;
  }
;

```

```

specification_statement:
  external_statement
  |
  intrinsic_statement
  |
  parameter_statement
  |
  dimension_statement
  |
  declaration_statement
  |
  save_statement
  |
  common_statement
  |
  equivalence_statement
  |
  implicit_statement
  |
  data_statement
  |
  namelist_statement
;

```

```

external_statement:
  RW_EXTERNAL external_list
;

```

```

external_list:
  external
  {
    $$ = merge( "{%s}", $1 );
  }
  |
  external_list ',' external
  {
    $$ = merge( "%s,%s", $1, $3 );
  }
;

```

```

external:
  IDENTIFIER
  {
    $$ = $1;
  }
;

```

```

intrinsic_statement:
  RW_INTRINSIC intrinsic_list
;

```

```

intrinsic_list:
  intrinsic
  {
    $$ = merge( "{%s}", $1 );
  }
  |
  intrinsic_list ',' intrinsic
  {
    $$ = merge( "%s,%s", $1, $3 );
  }
;

```

```

intrinsic:
  IDENTIFIER
  {
    $$ = $1;
  }
;

parameter_statement:
  RW_PARAMETER '(' parameter_list ')'
;

parameter_list:
  parameter
  {
    $$ = merge( "${%s}", $1 );
  }
  parameter_list ',' parameter
  {
    $$ = merge( "${%s}%s", $1, $3 );
  }
;

parameter:
  IDENTIFIER '=' expression
  {
    $$ = merge( "${%s}=${%s}", $1, $3 );
  }
;

dimension_statement:
  RW_DIMENSION dimension_list
;

dimension_list:
  dimension
  {
    $$ = merge( "${%s}", $1 );
  }
  dimension_list ',' dimension
  {
    $$ = merge( "${%s}%s", $1, $3 );
  }
;

dimension:
  IDENTIFIER '(' subscript_list ')'
  {
    $$ = merge( "${%s}(${%s})", $1, $3 );
  }
;

subscript_list:
  subscript
  {
    $$ = merge( "${%s}", $1 );
  }
  subscript_list ',' subscript
  {
    $$ = merge( "${%s}%s", $1, $3 );
  }
;

subscript:
  upper_bound
  {
    $$ = $1;
  }
  lower_bound ':' upper_bound

```

```

    {
        $$ = merge( "%s:%s", $1, $3 );
    }
;

lower_bound:
    expression
    {
        $$ = $1;
    }
;

upper_bound:
    expression
    {
        $$ = $1;
    }
    |
    upper_bound_adjustable
    {
        $$ = $1;
    }
;

upper_bound_adjustable:
    '*'
    {
        $$ = duplicate( "*" );
    }
;

declaration_statement:
    type declaration_list
;

declaration_list:
    declaration
    {
        $$ = merge( "(%s)", $1 );
    }
    |
    declaration_list ',' declaration
    {
        $$ = merge( "%s(%s)", $1, $3 );
    }
;

declaration:
    IDENTIFIER
    {
        $$ = merge( "(%s)", $1 );
    }
    |
    IDENTIFIER '(' subscript_list ')'
    {
        $$ = merge( "%s(%s)", $1, $3 );
    }
;

type:
    type_name optional_type_length
    {
        $$ = type( $1, $2 );
    }
;

type_name:
    RW_CHARACTER
    {
        $$ = duplicate( "CHARACTER" );
    }
    |
    RW_COMPLEX

```

```

    {
        $$ = duplicate( "COMPLEX" );
    }
|
    RW_DOUBLE_PRECISION
    {
        $$ = duplicate( "DOUBLE_PRECISION" );
    }
|
    RW_INTEGER
    {
        $$ = duplicate( "INTEGER" );
    }
|
    RW_LOGICAL
    {
        $$ = duplicate( "LOGICAL" );
    }
|
    RW_REAL
    {
        $$ = duplicate( "REAL" );
    }
|
    RW_UNDEFINED
    {
        $$ = duplicate( "UNDEFINED" );
    }
;

```

optional\_type\_length:

```

/* NULL */
{
    $$ = 0;
}
|
    type_length
    {
        $$ = $1;
    }
;

```

type\_length:

```

    '*' INTEGER
    {
        $$ = $2;
    }
|
    '*' type_length_adjustable
    {
        $$ = $2;
    }
;

```

type\_length\_adjustable:

```

    '(' '*' ')'
    {
        $$ = duplicate( "(" ) ;
    }
;

```

save\_statement:

```

    RW_SAVE optional_save_list
;

```

optional\_save\_list:

```

/* NULL */
{
    $$ = 0;
}
|
    save_list
    {
        $$ = $1;
    }
;

```

```

save_list:
  save
  {
    $$ = merge( "{%s}", $1 );
  }
  |
  save_list ',' save
  {
    $$ = merge( "%s{%s}", $1, $3 );
  }
  ;

save:
  IDENTIFIER
  {
    $$ = $1;
  }
  |
  common_name
  {
    $$ = $1;
  }
  ;

common_statement:
  RW_COMMON optional_common_name common_variable_list
  {
    common_name = 0;
  }
  ;

optional_common_name:
  /* NULL */
  {
    $$ = common_name = 0;
  }
  |
  common_name
  {
    $$ = common_name = $1;
  }
  ;

common_name:
  '/' optional_identifier '/'
  {
    $$ = $2;
  }
  ;

optional_identifier:
  /* NULL */
  {
    $$ = 0;
  }
  |
  IDENTIFIER
  {
    $$ = $1;
  }
  ;

common_variable_list:
  common_variable
  {
    $$ = merge( "{%s}", $1 );
  }
  |
  common_variable_list ',' common_variable
  {
    $$ = merge( "%s{%s}", $1, $3 );
  }
  ;

```

```

common_variable:
  IDENTIFIER
  {
    add_list( &clist, common_name, $1 );
    $$ = merge( "{%s}", $1 );
  }
  |
  IDENTIFIER '(' subscript_list ')'
  {
    add_list( &clist, common_name, $1 );
    $$ = merge( "{%s}{%s}", $1, $3 );
  }
;

```

```

equivalence_statement:
  RW_EQUIVALENCE equivalence_list
;

```

```

equivalence_list:
  equivalence
  {
    $$ = merge( "{%s}", $1 );
  }
  |
  equivalence_list ',' equivalence
  {
    $$ = merge( "%s{%s}", $1, $3 );
  }
;

```

```

equivalence:
  '(' variable_list ')'
  {
    $$ = $2;
  }
;

```

```

variable_list:
  variable
  {
    $$ = merge( "{%s}", $1 );
  }
  |
  variable_list ',' variable
  {
    $$ = merge( "%s{%s}", $1, $3 );
  }
;

```

```

implicit_statement:
  RW_IMPLICIT type '(' implicit_list ')'
;

```

```

implicit_list:
  implicit
  {
    $$ = merge( "{%s}", $1 );
  }
  |
  implicit_list ',' implicit
  {
    $$ = merge( "%s{%s}", $1, $3 );
  }
;

```

```

implicit:
  IDENTIFIER
  {
    $$ = merge( "{%s}", $1 );
  }
  |
  IDENTIFIER '-' IDENTIFIER

```

```

    {
        $$ = merge( "{%s}{%s}", $1, $3 );
    }
;

namelist_statement:
    RW_NAMELIST namelist_name namelist_list
;

namelist_name:
    '/' IDENTIFIER '/'
    {
        $$ = $2;
    }
;

namelist_list:
    namelist
    {
        $$ = merge( "{%s}", $1 );
    }
    |
    namelist_list ',' namelist
    {
        $$ = merge( "%s{%s}", $1, $3 );
    }
;

namelist:
    IDENTIFIER
    {
        $$ = $1;
    }
;

data_statement:
    RW_DATA data_list
;

data_list:
    data
    {
        $$ = merge( "{%s}", $1 );
    }
    |
    data_list optional_comma data
    {
        $$ = merge( "%s{%s}", $1, $3 );
    }
;

data:
    data_variable_list '/' data_constant_list '/'
    {
        $$ = merge( "{%s}{%s}", $1, $3 );
    }
;

data_variable_list:
    data_variable
    {
        $$ = merge( "{%s}", $1 );
    }
    |
    data_variable_list ',' data_variable
    {
        $$ = merge( "%s{%s}", $1, $3 );
    }
;

data_variable:
    variable

```

```

    {
        $$ = $1;
    }
    |
    data_implied_do_list
    {
        $$ = $1;
    }
    ;

data_implied_do_list:
    '(' data_variable_list ',' IDENTIFIER '=' expression_list ')'
    {
        $$ = merge "( %s, %s = %s )", list( $2, ",", " ), $4, list( $6, ",", " ) );
    }
    ;

data_constant_list:
    data_constant
    {
        $$ = merge( "{%s}", $1 );
    }
    |
    data_constant_list ',' data_constant
    {
        $$ = merge( "%s{%s}", $1, $3 );
    }
    ;

data_constant:
    data_initialization
    {
        $$ = $1;
    }
    |
    IDENTIFIER '*' data_initialization
    {
        $$ = merge( "%s * %s", $1, $3 );
    }
    |
    INTEGER '*' data_initialization
    {
        $$ = replicate( atoi( $1 ), $3, "{" );
    }
    ;

data_initialization:
    IDENTIFIER
    {
        $$ = $1;
    }
    |
    character_constant
    {
        $$ = $1;
    }
    |
    logical_constant
    {
        $$ = $1;
    }
    |
    signed_numerical_constant
    {
        $$ = $1;
    }
    ;

signed_numerical_constant:
    numerical_constant
    {
        $$ = $1;
    }
    |
    '+' numerical_constant %prec SIGN
    {

```



```

        $$ = merge( "+%s", $2 );
    }
    | '-' numerical_constant %prec SIGN
    {
        $$ = merge( "-%s", $2 );
    }
;

```

```

expression:
    parenthesis_expression
    {
        $$ = $1;
    }
    |
    simple_expression
    {
        $$ = $1;
    }
;

```

```

parenthesis_expression:
    '(' expression ')'
    {
        $$ = merge( "( %s )", $2 );
    }
;

```

```

simple_expression:
    variable
    {
        $$ = $1;
    }
    |
    constant
    {
        $$ = $1;
    }
    |
    arithmetic_expression
    {
        $$ = $1;
    }
    |
    character_expression
    {
        $$ = $1;
    }
    |
    relational_expression
    {
        $$ = $1;
    }
    |
    logical_expression
    {
        $$ = $1;
    }
    |
    unary_expression
    {
        $$ = $1;
    }
;

```

```

variable:
    IDENTIFIER
    {
        usage_list( clist, $1 );
        $$ = $1;
    }
    |
    IDENTIFIER string_subset
    {
        usage_list( clist, $1 );
    }
;

```

```

        $$ = merge( "%s%s", $1, $2 );
    }
    |
    array
    {
        $$ = $1;
    }
    ;

array:
IDENTIFIER '(' optional_expression_list ')'
{
    usage_list( clist, $1 );
    $$ = merge( "%s(%s)", $1, list( $3, " ", " ) );
}
|
IDENTIFIER '(' optional_expression_list ')' string_subset
{
    usage_list( clist, $1 );
    $$ = merge( "%s(%s)%s", $1, list( $3, " ", " ), $5 );
}
;

optional_expression_list:
/* NULL */
{
    $$ = 0;
}
|
expression_list
{
    $$ = $1;
}
;

expression_list:
expression
{
    $$ = merge( "{%s}", $1 );
}
|
expression_list ',' expression
{
    $$ = merge( "%s{%s}", $1, $3 );
}
;

string_subset:
IDENTIFIER ':' optional_expression ':' optional_expression ')'
{
    $$ = merge( "( %s : %s )", $2, $4 );
}
;

optional_expression:
/* NULL */
{
    $$ = 0;
}
|
expression
{
    $$ = $1;
}
;

constant:
character_constant
{
    $$ = $1;
}
|
logical_constant

```

```

    {
        $$ = $1;
    }
    |
    numerical_constant
    {
        $$ = $1;
    }
    ;

character_constant:
    HOLLERITH
    {
        $$ = $1;
    }
    |
    STRING
    {
        $$ = $1;
    }
    ;

logical_constant:
    RW_FALSE
    {
        $$ = duplicate( ".FALSE." );
    }
    |
    RW_TRUE
    {
        $$ = duplicate( ".TRUE." );
    }
    ;

numerical_constant:
    DOUBLE_PRECISION
    {
        $$ = $1;
    }
    |
    INTEGER
    {
        $$ = $1;
    }
    |
    REAL
    {
        $$ = $1;
    }
    ;

arithmetic_expression:
    expression '+' expression %prec '+'
    {
        $$ = merge( "%s + %s", $1, $3 );
    }
    |
    expression '-' expression %prec '-'
    {
        $$ = merge( "%s - %s", $1, $3 );
    }
    |
    expression '*' expression %prec '*'
    {
        $$ = merge( "%s * %s", $1, $3 );
    }
    |
    expression '/' expression %prec '/'
    {
        $$ = merge( "%s / %s", $1, $3 );
    }
    |
    expression EXPONENTIATE expression %prec EXPONENTIATE
    {
        $$ = merge( "%s ** %s", $1, $3 );
    }
    ;

```

```

character_expression:
    expression '/' '/' expression %prec CONCATENATE
    {
        $$ = merge( "%s // %s", $1, $4 );
    }
;

```

```

relational_expression:
    expression RW_EQ expression %prec RW_EQ
    {
        $$ = merge( "%s .EQ. %s", $1, $3 );
    }
    |
    expression RW_NE expression %prec RW_NE
    {
        $$ = merge( "%s .NE. %s", $1, $3 );
    }
    |
    expression RW_LT expression %prec RW_LT
    {
        $$ = merge( "%s .LT. %s", $1, $3 );
    }
    |
    expression RW_LE expression %prec RW_LE
    {
        $$ = merge( "%s .LE. %s", $1, $3 );
    }
    |
    expression RW_GT expression %prec RW_GT
    {
        $$ = merge( "%s .GT. %s", $1, $3 );
    }
    |
    expression RW_GE expression %prec RW_GE
    {
        $$ = merge( "%s .GE. %s", $1, $3 );
    }
;

```

```

logical_expression:
    expression RW_AND expression %prec RW_AND
    {
        $$ = merge( "%s .AND. %s", $1, $3 );
    }
    |
    expression RW_OR expression %prec RW_OR
    {
        $$ = merge( "%s .OR. %s", $1, $3 );
    }
    |
    expression RW_EQV expression %prec RW_EQV
    {
        $$ = merge( "%s .EQV. %s", $1, $3 );
    }
    |
    expression RW_NEQV expression %prec RW_NEQV
    {
        $$ = merge( "%s .NEQV. %s", $1, $3 );
    }
;

```

```

unary_expression:
    '+' expression %prec SIGN
    {
        $$ = merge( "+%s", $2 );
    }
    |
    '-' expression %prec SIGN
    {
        $$ = merge( "-%s", $2 );
    }
    |
    RW_NOT expression %prec RW_NOT
    {
        $$ = merge( ".NOT. %s", $2 );
    }
;

```

```

;

executable_statement:
    do_statement
    |
    logical_if_statement
    |
    block_if_statement
    |
    else_statement
    |
    else_if_statement
    |
    end_if_statement
    |
    subset_executable_statement
;

do_statement:
    RW_DO optional_integer IDENTIFIER '=' expression_list
;

optional_integer:
    /* NULL */
    {
        $$ = 0;
    }
    |
    INTEGER
    {
        $$ = $1;
    }
;

logical_if_statement:
    if_expression subset_executable_statement
;

if_expression:
    RW_IF '(' expression ')'
;

block_if_statement:
    RW_IF '(' expression ')' RW_THEN
;

else_statement:
    RW_ELSE
;

else_if_statement:
    RW_ELSE_IF '(' expression ')' RW_THEN
;

end_if_statement:
    RW_END_IF
;

subset_executable_statement:
    assignment_statement
    |
    assign_statement
    |
    arithmetic_if_statement
    |
    continue_statement
    |
    call_statement
    |
    return_statement

```

```

    unconditional_go_to_statement
  |
    computed_go_to_statement
  |
    assigned_go_to_statement
  |
    stop_statement
  |
    pause_statement
  |
    io_statement
  ;

assignment_statement:
    variable '=' expression
  ;

assign_statement:
    RW_ASSIGN INTEGER RW_TO IDENTIFIER
  ;

arithmetic_if_statement:
    RW_IF '(' expression ')' integer_list
  ;

continue_statement:
    RW_CONTINUE
  ;

call_statement:
    RW_CALL IDENTIFIER
  |
    RW_CALL IDENTIFIER optional_actual_argument_list
  ;

optional_actual_argument_list:
    '(' ')'
    {
        $$ = 0;
    }
  |
    '(' actual_argument_list ')'
    {
        $$ = $2;
    }
  ;

actual_argument_list:
    actual_argument
    {
        $$ = merge( "%s", $1 );
    }
  |
    actual_argument_list ',' actual_argument
    {
        $$ = merge( "%s(%s)", $1, $3 );
    }
  ;

actual_argument:
    expression
    {
        $$ = $1;
    }
  |
    actual_argument_alternate_return
    {
        $$ = $1;
    }
  ;

actual_argument_alternate_return:

```

```

    '*' INTEGER
    {
        $$ = merge( "%s", $2 );
    }
;

return_statement:
    RW_RETURN optional_expression
;

unconditional_go_to_statement:
    RW_GO_TO INTEGER
;

computed_go_to_statement:
    RW_GO_TO '(' integer_list ')' optional_comma expression
;

assigned_go_to_statement:
    RW_GO_TO IDENTIFIER
    |
    RW_GO_TO IDENTIFIER optional_comma '(' integer_list ')'
;

optional_comma:
    /* NULL */
    |
    ','
;

integer_list:
    INTEGER
    {
        $$ = merge( "%s", $1 );
    }
    |
    integer_list ',' INTEGER
    {
        $$ = merge( "%s%s", $1, $3 );
    }
;

pause_statement:
    RW_PAUSE optional_expression
;

stop_statement:
    RW_STOP optional_expression
;

io_statement:
    open_statement
    |
    close_statement
    |
    inquire_statement
    |
    read_statement
    |
    write_statement
    |
    print_statement
    |
    backspace_statement
    |
    rewind_statement
    |
    endfile_statement
;

open_statement:

```

```

        RW_OPEN '(' control_information_list ')'
    ;

close_statement:
    RW_CLOSE '(' control_information_list ')'
    ;

inquire_statement:
    RW_INQUIRE '(' control_information_list ')'
    ;

read_statement:
    RW_READ '(' control_information_list ')' optional_io_list
    |
    RW_READ control
    |
    RW_READ control ',' io_list
    ;

write_statement:
    RW_WRITE '(' control_information_list ')' optional_io_list
    ;

print_statement:
    RW_PRINT control
    |
    RW_PRINT control ',' io_list
    ;

backspace_statement:
    RW_BACKSPACE '(' control_information_list ')'
    |
    RW_BACKSPACE control
    ;

rewind_statement:
    RW_REWIND '(' control_information_list ')'
    |
    RW_REWIND control
    ;

endfile_statement:
    RW_ENDFILE '(' control_information_list ')'
    |
    RW_ENDFILE control
    ;

control_information_list:
    control_information
    {
        $$ = merge( "%s)", $1 );
    }
    |
    control_information_list ',' control_information
    {
        $$ = merge( "%s(%s)", $1, $3 );
    }
    ;

control_information:
    control
    {
        $$ = $1;
    }
    |
    IDENTIFIER '=' control
    {
        $$ = merge( "%s = %s", $1, $3 );
    }
    ;

```



```

control:
    variable
    {
        $$ = $1;
    }
    |
    constant
    {
        $$ = $1;
    }
    !
    '*'
    {
        $$ = duplicate( "*" );
    }
    ;

optional_io_list:
    /* NULL */
    {
        $$ = 0;
    }
    |
    io_list
    {
        $$ = $1;
    }
    ;

io_list:
    io
    {
        $$ = merge( "{%s}", $1 );
    }
    |
    io_list ',' io
    {
        $$ = merge( "%s{%s}", $1, $3 );
    }
    ;

io:
    expression
    {
        $$ = $1;
    }
    |
    io_implied_do_list
    {
        $$ = $1;
    }
    ;

io_implied_do_list:
    '(' io_list ',' IDENTIFIER '=' expression_list ')'
    {
        $$ = merge( "( %s, %s = %s )", list( $2, ", ", " ), $4, list( $6, ", ", " ) );
    }
    ;

format_statement:
    RW_FORMAT
    ;

%%

FILE: common/include/list.h

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory

```

```

* Author:  Stephen R. Wachtel
*/

#define LIST struct list_type
LIST
{
    char *identifier;
    char *alternate;
    int usage;
    LIST *next;
};

extern LIST *end_list( );
extern LIST *add_list( );
extern LIST *find_list( );
extern void usage_list( );
extern LIST *find_index( );
extern void print_list( );
extern void delete_list( );

FILE: common/library/Makefile

#
# Copyright 1991
# Georgia Institute of Technology
# Computer Engineering Research Laboratory
# Author:  Stephen R. Wachtel
#

CC = cc -g
INCLUDE = ../include
CFLAGS = -I$(INCLUDE)
LIBRARY = library.a

OBJECTS = \
    count.o \
    duplicate.o \
    hollerith.o \
    link_list.o \
    list.o \
    main.o \
    merge.o \
    non_blank.o \
    parse.o \
    replicate.o \
    type.o \
    uppercase.o \
    yyerror.o \
    yygetc.o \
    yywrap.o

$(LIBRARY): $(OBJECTS)
    ar crv $(LIBRARY) $(OBJECTS)
    ranlib $(LIBRARY)

.SUFFIXES: .c .o
.c.o:
    $(CC) -c $(CFLAGS) $<

clean:
    rm -f $(LIBRARY) $(OBJECTS)

FILE: common/library/count.c

/*
* Copyright 1991
* Georgia Institute of Technology
* Computer Engineering Research Laboratory
* Author:  Stephen R. Wachtel
*/

```

```

int count( string, length, c )
register char *string;
register int length;
register char c;
{
    register int c_count = 0;

    while ( length != 0 )
    {
        if ( *string == c )
            c_count++;

        string++;
        length--;
    }

    return( c_count );
} /* count */

```

FILE: common/library/duplicate.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include <stdio.h>
#include <string.h>
#include <malloc.h>

char *duplicate( string )
register char *string;
{
    register char *temporary = (char *)NULL;

    if ( string != (char *)NULL )
    {
        if ( (temporary = (char *)malloc( strlen( string ) + 1 ) ) != (char *)NULL )
            strcpy( temporary, string );
        else
            fprintf( stderr, "ERROR: duplicate( %s )\n", string );
    }

    return( temporary );
} /* duplicate */

```

FILE: common/library/hollerith.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include <stdio.h>

char *hollerith( string, delimiter )
register char *string;
register char delimiter;
{
    int hollerith_length;
    register int string_length = 0;

    sscanf( string, "%dh", &hollerith_length );

    string[ string_length++ ] = delimiter;
    while ( hollerith_length != 0 )
    {

```

```

    if ( ( string[ string_length ] = yyinput( ) ) == '\n' )
    {
        yyunput( string[ string_length ] );
        break;
    }

    string_length++;
    hollerith_length--;
}
string[ string_length++ ] = delimiter;

string[ string_length ] = '\0';
return( string );
} /* hollerith */

```

FILE: common/library/link\_list.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include <stdio.h>
#include <string.h>
#include <malloc.h>
#include "list.h"

extern FILE *yyin;
extern FILE *yyout;
extern char *merge( );

LIST *end_list( list )
register LIST *list;
{
    if ( list != (LIST *)NULL )
    {
        while ( list->next != (LIST *)NULL )
            list = list->next;
    }

    return( list );
} /* end_list */

LIST *add_list( list, common_name, identifier )
register LIST **list;
register char *common_name;
register char *identifier;
{
    register LIST *temporary = (LIST *)malloc( sizeof( LIST ) );

    if ( common_name == (char *)NULL )
    {
        temporary->identifier = identifier;
        temporary->alternate = (char *)NULL;
    }
    else
    {
        temporary->identifier = identifier;
        temporary->alternate = merge( "%s.%s", common_name, identifier );
    }

    temporary->usage = 0;
    temporary->next = (LIST *)NULL;

    if ( *list == (LIST *)NULL )
        *list = temporary;
    else
        end_list( *list )->next = temporary;

    return( temporary );
} /* add_list */

```

```

LIST *find_list( list, identifier )
register LIST *list;
register char *identifier;
{
    while ( list != (LIST *)NULL )
    {
        if ( strcmp( list->identifier, identifier ) == 0 )
            return( list );

        list = list->next;
    }

    return( (LIST *)NULL );
} /* find_list */

void usage_list( list, identifier )
register LIST *list;
register char *identifier;
{
    register LIST *temporary;

    if ( ( temporary = find_list( list, identifier ) ) != (LIST *)NULL )
        temporary->usage++;
} /* usage_list */

LIST *find_index( list, index )
register LIST *list;
register int index;
{
    while ( list != (LIST *)NULL )
    {
        if ( --index == 0 )
            return( list );

        list = list->next;
    }

    return( (LIST *)NULL );
} /* find_index */

void print_list( list, name )
register LIST *list;
register char *name;
{
    while ( list != (LIST *)NULL )
    {
        fprintf( yyout, "d(\\UU%s.FOR\\", \"%s\\", \"%s\\", %d)\\n", name, list->identifier,
list->alternate, list->usage );

        list = list->next;
    }
} /* print_list */

void delete_list( list )
register LIST *list;
{
    if ( list != (LIST *)NULL )
    {
        delete_list( list->next );

        free( list );
    }
} /* delete_list */

FILE: common/library/lis~.c

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

extern char *parse( );

```

```

extern char *merge( );

char *list( input_list, delimiter )
register char *input_list;
register char *delimiter;
{
    register char *output_list;
    register char *list;
    register char *temporary;

    output_list = parse( input_list );
    list = parse( input_list );

    while ( list != (char *)0 )
    {
        temporary = merge( "%s%s%s", output_list, delimiter, list );

        free( output_list );
        free( list );

        output_list = temporary;
        list = parse( input_list );
    }

    return( output_list );
} /* list */

FILE: common/library/main.c

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include <stdio.h>

extern FILE *yyin;
extern FILE *yyout;

#define PROGRAM argument[ 0 ]
#define INPUT_FILE argument[ 1 ]
#define OUTPUT_FILE argument[ 2 ]

int main( number_argument, argument )
int number_argument;
char *argument[ ];
{
    if ( number_argument == 1 )
    {
        yyin = stdin;
        yyout = stdout;

        yyparse( );
        exit( 0 );
    }

    if ( number_argument == 3 )
    {
        if ( ( yyin = fopen( INPUT_FILE, "r" ) ) == (FILE *)NULL )
        {
            fprintf( stderr, "%s: ERROR - unable to open input file '%s'\n", PROGRAM,
INPUT_FILE );
            exit( -1 );
        }

        if ( ( yyout = fopen( OUTPUT_FILE, "w" ) ) == (FILE *)NULL )
        {
            fprintf( stderr, "%s: ERROR - unable to open output file '%s'\n", PROGRAM,
OUTPUT_FILE );
            exit( -1 );
        }
    }
}

```

```

    yyparse( );
    exit( 0 );
}

fprintf( stderr, "usage: %s <input file> <output file>\n", PROGRAM );
exit( 0 );
} /* main */

FILE: common/library/merge.c

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include <stdio.h>
#include <string.h>
#include <malloc.h>

#define STRLEN( s ) ( strlen( s ) - 2 )

char *merge( string, a, b, c, d )
register char *string;
register char *a;
register char *b;
register char *c;
register char *d;
{
    register char *temporary = (char *)NULL;

    switch ( count( string, strlen( string ), '%' ) )
    {
        case 0:
            if ( ( temporary = (char *)malloc( strlen( string ) + 1 ) ) != (char *)NULL )
                sprintf( temporary, string );
            else
                fprintf( stderr, "ERROR: merge( %s )\n", string );
            break;

        case 1:
            if ( ( temporary = (char *)malloc( strlen( string ) + STRLEN( a ) + 1 ) ) !=
(char *)NULL )
                sprintf( temporary, string, a );
            else
                fprintf( stderr, "ERROR: merge( %s, %s )\n", string, a );
            break;

        case 2:
            if ( ( temporary = (char *)malloc( strlen( string ) + STRLEN( a ) + STRLEN( b
) + 1 ) ) != (char *)NULL )
                sprintf( temporary, string, a, b );
            else
                fprintf( stderr, "ERROR: merge( %s, %s, %s )\n", string, a, b );
            break;

        case 3:
            if ( ( temporary = (char *)malloc( strlen( string ) + STRLEN( a ) + STRLEN( b
) + STRLEN( c ) + 1 ) ) != (char *)NULL )
                sprintf( temporary, string, a, b, c );
            else
                fprintf( stderr, "ERROR: merge( %s, %s, %s, %s )\n", string, a, b, c );
            break;

        case 4:
            if ( ( temporary = (char *)malloc( strlen( string ) + STRLEN( a ) + STRLEN( b
) + STRLEN( c ) + STRLEN( d ) + 1 ) ) != (char *)NULL )
                sprintf( temporary, string, a, b, c, d );
            else
                fprintf( stderr, "ERROR: merge( %s, %s, %s, %s, %s )\n", string, a, b, c, d
);
            break;

        default:
            fprintf( stderr, "ERROR: merge( %s )\n", string );
    }
}

```

```

        break;
    }

    return( temporary );
} /* merge */

```

FILE: common/library/non\_blank.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include <string.h>

char *non_blank( string )
register char *string;
{
    register int offset;
    register int length;

    length = strlen( string ) - 1;
    while ( ( string[ length ] == ' ' ) && ( string[ length ] != '\0' ) )
        string[ length-- ] = '\0';

    offset = 0;
    while ( ( string[ offset ] == ' ' ) && ( string[ offset ] != '\0' ) )
        string[ offset++ ] = '\0';

    strcpy( string, &string[ offset ] );

    if ( strlen( string ) != 0 )
        return( string );
    else
        return( 0 );
} /* non_blank */

```

FILE: common/library/parse.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include <string.h>

extern char *duplicate( );

char *parse( list )
register char *list;
{
    register int length = 0;
    register int brace = 0;
    register char *temporary = (char *)0;

    for (;;)
    {
        switch ( list[ length ] )
        {
            case '(':
                brace++;
                break;

            case ')':
                brace--;
                break;
        }
    }
}

```



```

        if ( brace == 0 )
            break;

        length++;
    }

    if ( length != 0 )
    {
        list[ length ] = '\0';
        temporary = duplicate( list + 1 );
        strcpy( list, list + 1 + length );
    }
    else
    {
        if ( list[ length ] != '\0' )
        {
            temporary = duplicate( list );
            list[ length ] = '\0';
        }
    }

    return temporary;
} /* parse */

```

FILE: common/library/replicate.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```

#include <string.h>
#include <malloc.h>

```

```

char *replicate( count, string, delimiter )
register int count;
register char *string;
register char *delimiter;
{
    register char *temporary = (char *)malloc( ( count * strlen( string ) ) + ( ( count -
1 ) * strlen( delimiter ) ) + 1 );

    if ( temporary != (char *)0 )
    {
        strcpy( temporary, string );

        while ( --count != 0 )
        {
            strcat( temporary, delimiter );
            strcat( temporary, string );
        }

        return( temporary );
    }
} /* replicate */

```

FILE: common/library/type.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```

extern char *merge( );

```

```

char *type( type_name, optional_type_length )
register char *type_name;
register char *optional_type_length;
{

```

```

    if ( optional_type_length != 0 )
        return( merge( "%s%s", type_name, optional_type_length ) );
    else
        return( type_name );
} /* type */

```

FILE: common/library/uppercase.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

char *uppercase( string )
register char *string;
{
    register int index = 0;

    while ( string[ index ] != '\0' )
    {
        string[ index ] = toupper( string[ index ] );
        index++;
    }

    return( string );
} /* uppercase */

```

FILE: common/library/yyerror.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include <stdio.h>

extern int yylineno;

void yyerror( string )
register char *string;
{
    fprintf( stderr, "line %d, %s\n", yylineno, string );
    exit( -1 );
} /* yyerror */

```

FILE: common/library/yygetc.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include <stdio.h>
#include <ctype.h>

extern int yylineno;

int tab( length )
register int length;
{
    while ( length-- != 0 )

```

```

        yyunput( ' ' );

        return( ' ' );
    } /* tab */

int yygetc( file )
register FILE *file;
{
    int c;
    int column[ 6 ];

loop:
    if ( ( c = getc( file ) ) == '\t' )
        c = tab( 6 );

    if ( c != '\n' )
        return( c );

    if ( ( column[ 0 ] = getc( file ) ) != ' ' )
        goto abort_0;
    if ( ( column[ 1 ] = getc( file ) ) != ' ' )
        goto abort_1;
    if ( ( column[ 2 ] = getc( file ) ) != ' ' )
        goto abort_2;
    if ( ( column[ 3 ] = getc( file ) ) != ' ' )
        goto abort_3;
    if ( ( column[ 4 ] = getc( file ) ) != ' ' )
        goto abort_4;
    if ( isspace( column[ 5 ] = getc( file ) ) )
        goto abort_5;

    yylineno++;
    goto loop;

abort_5:
    if ( column[ 5 ] == '\t' )
        tab( 1 );
    else
    {
        yyunput( column[ 5 ] );
        if ( column[ 5 ] == '\n' )
            yylineno++;
    }

abort_4:
    if ( column[ 4 ] == '\t' )
        tab( 2 );
    else
    {
        yyunput( column[ 4 ] );
        if ( column[ 4 ] == '\n' )
            yylineno++;
    }

abort_3:
    if ( column[ 3 ] == '\t' )
        tab( 3 );
    else
    {
        yyunput( column[ 3 ] );
        if ( column[ 3 ] == '\n' )
            yylineno++;
    }

abort_2:
    if ( column[ 2 ] == '\t' )
        tab( 4 );
    else
    {
        yyunput( column[ 2 ] );
        if ( column[ 2 ] == '\n' )
            yylineno++;
    }

abort_1:
    if ( column[ 1 ] == '\t' )
        tab( 5 );
    else
    {
        yyunput( column[ 1 ] );

```

```

        if ( column[ 1 ] == '\n' )
            yylineno++;
    }
abort_0:
    if ( column[ 0 ] == '\t' )
        tab( 6 );
    else
    {
        yyunput( column[ 0 ] );
        if ( column[ 0 ] == '\n' )
            yylineno++;
    }

    return( c );
} /* yygetc */

```

FILE: common/library/yywrap.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

int yywrap( )
{
    return( 1 );
} /* yywrap */

```

FILE: common/scanner.l

```

%{
/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */
%}

%a 10000
%e 10000
%k 10000
%n 10000
%o 10000
%p 10000

a  [aA]
b  [bB]
c  [cC]
d  [dD]
e  [eE]
f  [fF]
g  [gG]
h  [hH]
i  [iI]
j  [jJ]
k  [kK]
l  [lL]
m  [mM]
n  [nN]
o  [oO]
p  [pP]
q  [qQ]
r  [rR]
s  [sS]
t  [tT]
u  [uU]
v  [vV]
w  [wW]
x  [xX]
y  [yY]

```

```
z {z2}
```

```
%{
#include "grammar.h"
extern char *yylval;
```

```
#undef YYLMAX
#define YYLMAX (256*20)
```

```
extern char *duplicate( );
extern char *hollerith( );
extern char *non_blank( );
extern char *uppercase( );
%}
```

```
%%
```

```
^[\\*cC].*[\\n] |
^[\\ ]*[\\n] {
#ifdef DEBUG
    ECHO;
#endif
    yylval = duplicate( yytext );
    return( COMMENT );
}
```

```
[\\ ] {
#ifdef DEBUG
    ECHO;
#endif
    /* return( '\\ ' ) */;
}
```

```
[\\&] {
#ifdef DEBUG
    ECHO;
#endif
    return( '\\&' );
}
```

```
[\\()] {
#ifdef DEBUG
    ECHO;
#endif
    return( '\\(' );
}
```

```
[\\)] {
#ifdef DEBUG
    ECHO;
#endif
    return( '\\)' );
}
```

```
[\\*] {
#ifdef DEBUG
    ECHO;
#endif
    return( '\\*' );
}
```

```
[\\*][\\*] {
#ifdef DEBUG
    ECHO;
#endif
    return( EXPONENTIATE );
}
```

```
[\\+] {
```

```

#ifdef DEBUG
    ECHO;
#endif
    return( '\\' );
}

[\\] {
#ifdef DEBUG
    ECHO;
#endif
    return( '\\, ' );
}

[\\-] {
#ifdef DEBUG
    ECHO;
#endif
    return( '\\-' );
}

[\\.] {
#ifdef DEBUG
    ECHO;
#endif
    return( '\\.' );
}

[\\/] {
#ifdef DEBUG
    ECHO;
#endif
    return( '\\/' );
}

[\\:] {
#ifdef DEBUG
    ECHO;
#endif
    return( '\\:' );
}

[\\=] {
#ifdef DEBUG
    ECHO;
#endif
    return( '\\=' );
}

[\\n] {
#ifdef DEBUG
    ECHO;
#endif
    /* return( '\\n' ) */;
}

[\\t] {
#ifdef DEBUG
    ECHO;
#endif
    /* return( '\\t' ) */;
}

[\\.]{a}{n}{d}[\\.] {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_AND );
}

[\\.]{e}{q}[\\.] {
#ifdef DEBUG

```

```

    ECHO;
#endif
    return( RW_EQ );
}

[\.]{e}{q}{v}{\.} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_EQV );
}

[\.]{f}{a}{l}{s}{e}{\.} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_FALSE );
}

[\.]{g}{e}{\.} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_GE );
}

[\.]{g}{t}{\.} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_GT );
}

[\.]{l}{e}{\.} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_LE );
}

[\.]{l}{t}{\.} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_LT );
}

[\.]{n}{e}{\.} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_NE );
}

[\.]{n}{e}{q}{v}{\.} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_NEQV );
}

[\.]{n}{o}{t}{\.} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_NOT );
}

[\.]{o}{r}{\.} {
#ifdef DEBUG
    ECHO;

```

```

#endif
    return( RW_OR );
}

[\.]{t}{r}{u}{e}[\.] {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_TRUE );
}

{a}{s}{s}{i}{g}{n} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_ASSIGN );
}

{b}{a}{c}{k}{s}{p}{a}{c}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_BACKSPACE );
}

{b}{l}{o}{c}{k}[\ ]*{d}{a}{t}{a} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_BLOCK_DATA );
}

{c}{a}{l}{l} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_CALL );
}

{c}{h}{a}{r}{a}{c}{t}{e}{r} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_CHARACTER );
}

{c}{l}{o}{s}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_CLOSE );
}

{c}{o}{m}{m}{o}{n} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_COMMON );
}

{c}{o}{m}{p}{l}{e}{x} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_COMPLEX );
}

{c}{o}{n}{t}{i}{n}{u}{e} {
#ifdef DEBUG
    ECHO;
#endif

```



```

        return( RW_CONTINUE );
    }

    {d}{a}{t}{a} {
    #ifdef DEBUG
        ECHO;
    #endif
        return( RW_DATA );
    }

    {d}{i}{m}{e}{n}{s}{i}{o}{n} {
    #ifdef DEBUG
        ECHO;
    #endif
        return( RW_DIMENSION );
    }

    {d}{o} {
    #ifdef DEBUG
        ECHO;
    #endif
        return( RW_DO );
    }

    {d}{o}{u}{b}{l}{e}{\ }*{p}{r}{e}{c}{i}{s}{i}{o}{n} {
    #ifdef DEBUG
        ECHO;
    #endif
        return( RW_DOUBLE_PRECISION );
    }

    {e}{l}{s}{e} {
    #ifdef DEBUG
        ECHO;
    #endif
        return( RW_ELSE );
    }

    {e}{l}{s}{e}{\ }*{i}{f} {
    #ifdef DEBUG
        ECHO;
    #endif
        return( RW_ELSE_IF );
    }

    {e}{n}{d} {
    #ifdef DEBUG
        ECHO;
    #endif
        return( RW_END );
    }

    {e}{n}{d}{\ }*{i}{f} {
    #ifdef DEBUG
        ECHO;
    #endif
        return( RW_END_IF );
    }

    {e}{n}{d}{f}{i}{l}{e} {
    #ifdef DEBUG
        ECHO;
    #endif
        return( RW_ENDFILE );
    }

    {e}{n}{t}{r}{y} {
    #ifdef DEBUG
        ECHO;
    #endif
        return( RW_ENTRY );
    }

```

```

    }

    {e}{q}{u}{i}{v}{a}{l}{l}{e}{n}{c}{e} {
#ifdef DEBUG
        ECHO;
#endif
        return( RW_EQUIVALENCE );
    }

    {e}{x}{t}{e}{r}{n}{a}{l} {
#ifdef DEBUG
        ECHO;
#endif
        return( RW_EXTERNAL );
    }

    {f}{o}{r}{m}{a}{t}.* {
#ifdef DEBUG
        ECHO;
#endif
        yyval = duplicate( yytext );
        return( RW_FORMAT );
    }

    {f}{u}{n}{c}{t}{i}{o}{n} {
#ifdef DEBUG
        ECHO;
#endif
        return( RW_FUNCTION );
    }

    {g}{o}{[ \ ]*(t){o} {
#ifdef DEBUG
        ECHO;
#endif
        return( RW_GO_TO );
    }

    {i}{f} {
#ifdef DEBUG
        ECHO;
#endif
        return( RW_IF );
    }

    {i}{m}{p}{l}{i}{c}{i}{t} {
#ifdef DEBUG
        ECHO;
#endif
        return( RW_IMPLICIT );
    }

    {i}{n}{c}{l}{u}{d}{e} {
#ifdef DEBUG
        ECHO;
#endif
        return( RW_INCLUDE );
    }

    {i}{n}{q}{u}{i}{r}{e} {
#ifdef DEBUG
        ECHO;
#endif
        return( RW_INQUIRE );
    }

    {i}{n}{t}{e}{g}{e}{r} {
#ifdef DEBUG
        ECHO;
#endif
        return( RW_INTEGER );
    }

```

```

    }

    {i}{n}{t}{r}{i}{n}{s}{i}{c} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_INTRINSIC );
}

```

```

    {l}{o}{g}{i}{c}{a}{l} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_LOGICAL );
}

```

```

    {n}{a}{m}{e}{l}{i}{s}{t} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_NAMELIST );
}

```

```

    {o}{p}{e}{n} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_OPEN );
}

```

```

    {p}{a}{r}{a}{m}{e}{t}{e}{r} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_PARAMETER );
}

```

```

    {p}{a}{u}{s}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_PAUSE );
}

```

```

    {p}{r}{i}{n}{t} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_PRINT );
}

```

```

    {p}{r}{o}{g}{r}{a}{m} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_PROGRAM );
}

```

```

    {r}{e}{a}{d} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_READ );
}

```

```

    {r}{e}{a}{l} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_REAL );
}

```

```

{r}{e}{t}{u}{r}{n} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_RETURN );
}

{r}{e}{w}{i}{n}{d} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_REWIND );
}

{s}{a}{v}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_SAVE );
}

{s}{t}{o}{p} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_STOP );
}

{s}{u}{b}{r}{o}{u}{t}{i}{n}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_SUBROUTINE );
}

{t}{h}{e}{n} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_THEN );
}

{t}{o} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_TO );
}

{w}{r}{i}{t}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_WRITE );
}

{u}{n}{d}{e}{f}{i}{n}{e}{d} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_UNDEFINED );
}

[a-zA-Z][a-zA-Z0-9]* {
#ifdef DEBUG
    ECHO;
#endif
    yy1val = duplicate( uppercase( yytext ) );
    return( IDENTIFIER );
}

```

```

^[0-9][0-9][0-9][0-9][0-9][\ ] {
#ifdef DEBUG
    ECHO;
#endif
    yylval = duplicate( non_blank( yytext ) );
    return( LABEL );
}

[0-9]+ |
[0-9]+/\.[a-zA-Z]+\.( {
#ifdef DEBUG
    ECHO;
#endif
    yylval = duplicate( yytext );
    return( INTEGER );
}

[0-9]+\.[0-9]*([eE][\+|-]?[0-9]+)? |
[0-9]*\.[0-9]+([eE][\+|-]?[0-9]+)? |
[0-9]+([eE][\+|-]?[0-9]+)? {
#ifdef DEBUG
    ECHO;
#endif
    yylval = duplicate( yytext );
    return( REAL );
}

[0-9]+\.[0-9]*([dD][\+|-]?[0-9]+)? |
[0-9]*\.[0-9]+([dD][\+|-]?[0-9]+)? |
[0-9]+([dD][\+|-]?[0-9]+)? {
#ifdef DEBUG
    ECHO;
#endif
    yylval = duplicate( yytext );
    return( DOUBLE_PRECISION );
}

\[^\']* \[^\"]* {
#ifdef DEBUG
    ECHO;
#endif
    yytext[ 0 ] = '\\"';
    yytext[ strlen( yytext ) - 1 ] = '\\"';
    yylval = duplicate( yytext );
    return( STRING );
}

[0-9]+[hH] {
#ifdef DEBUG
    ECHO;
#endif
    yylval = duplicate( hollerith( yytext, '\\"' ) );
    return( HOLLERITH );
}

```

## 10. Appendix E: ctimer program source

FILE: ctimer/Makefile

```
#
# Copyright 1991
# Georgia Institute of Technology
# Computer Engineering Research Laboratory
# Author: Stephen R. Wachtel
#

default:    ctimer

CC = cc -g
INCLUDE = include
CFLAGS = -I$(INCLUDE)
LIBRARY = statement/statement.a library/library.a

OBJECTS = \
    $(INCLUDE)/grammar.h \
    *grammar.[co] \
    *scanner.[co] \
    yytrace.[co] \
    y.output

PROGRAMS = \
    *ctimer

grammar.c: grammar.y
    yacc -dv grammar.y
    mv y.tab.h $(INCLUDE)/grammar.h
    mv y.tab.c grammar.c

scanner.c: scanner.l
    lex -vt scanner.l | sed 's/getc/yygetc/' >scanner.c

scanner.o: scanner.c $(INCLUDE)/grammar.h
    $(CC) $(CFLAGS) -c scanner.c

grammar.o: grammar.c
    $(CC) $(CFLAGS) -c grammar.c

ctimer: grammar.o scanner.o $(LIBRARY)
    $(CC) -o ctimer grammar.o scanner.o $(LIBRARY)

sgrammar.c: grammar.c yytoken.awk
    awk -f yytoken.awk <grammar.c >sgrammar.c

sgrammar.o: sgrammar.c
    $(CC) $(CFLAGS) -c sgrammar.c

sctimer: sgrammar.o scanner.o $(LIBRARY)
    $(CC) -o sctimer sgrammar.o scanner.o $(LIBRARY)

dscanner.c: scanner.c
    cp scanner.c dscanner.c

dscanner.o: dscanner.c $(INCLUDE)/grammar.h
    $(CC) $(CFLAGS) -DDEBUG -c dscanner.c

dctimer: grammar.o dscanner.o $(LIBRARY)
    $(CC) -o dctimer grammar.o dscanner.o $(LIBRARY)

tgrammar.c: grammar.c
    sed 's/yystack:/& yytrace(yystate);/' <grammar.c >tgrammar.c

tgrammar.o: tgrammar.c
    $(CC) $(CFLAGS) -c tgrammar.c
```

```
tctimer:  tgrammar.o scanner.o yytrace.o $(LIBRARY)
          $(CC) -o tctimer tgrammar.o scanner.o yytrace.o $(LIBRARY)
```

```
yytrace.c: grammar.c yytrace.awk
          awk -f yytrace.awk <y.output >yytrace.c
```

```
yytrace.o: yytrace.c
          $(CC) $(CFLAGS) -c yytrace.c
```

```
clean:
        cd statement; make clean
        cd library; make clean
        rm -f $(PROGRAMS) $(OBJECTS)
```

```
FILE: ctimer/grammar.y
```

```
/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */
```

```
/*
 * FORTRAN 77
 */
```

```
%token RW_AND
%token RW_ASSIGN
%token RW_BACKSPACE
%token RW_BLOCK_DATA
%token RW_CALL
%token RW_CHARACTER
%token RW_CLOSE
%token RW_COMMON
%token RW_COMPLEX
%token RW_CONTINUE
%token RW_DATA
%token RW_DIMENSION
%token RW_DO
%token RW_DOUBLE_PRECISION
%token RW_ELSE
%token RW_ELSE_IF
%token RW_END
%token RW_END IF
%token RW_ENDFILE
%token RW_ENTRY
%token RW_EQ
%token RW_EQUIVALENCE
%token RW_EQV
%token RW_EXTERNAL
%token RW_FALSE
%token RW_FORMAT
%token RW_FUNCTION
%token RW_GE
%token RW_GO_TO
%token RW_GT
%token RW_IF
%token RW_IMPLICIT
%token RW_INCLUDE
%token RW_INQUIRE
%token RW_INTEGER
%token RW_INTRINSIC
%token RW_LE
%token RW_LOGICAL
%token RW_LT
%token RW_NAMELIST
%token RW_NE
%token RW_NEQV
%token RW_NOT
%token RW_OPEN
%token RW_OR
%token RW_PARAMETER
%token RW_PAUSE
%token RW_PRINT
```

```

%token RW_PROGRAM
%token RW_READ
%token RW_REAL
%token RW_RETURN
%token RW_REWIND
%token RW_SAVE
%token RW_STOP
%token RW_SUBROUTINE
%token RW_THEN
%token RW_TO
%token RW_TRUE
%token RW_WRITE
%token RW_UNDEFINED

%token COMMENT
%token CONCATENATE
%token DOUBLE_PRECISION
%token EXPONENTIATE
%token HOLLERITH
%token IDENTIFIER
%token INTEGER
%token LABEL
%token REAL
%token STRING

%left ','
%nonassoc ':'
%right '='
%left RW_EQV RW_NEQV
%left RW_OR
%left RW_AND
%left RW_NOT
%nonassoc RW_EQ RW_NE RW_LT RW_LE RW_GT RW_GE
%left CONCATENATE
%left '+' '-'
%left '*' '/'
%right EXPONENTIATE
%left SIGN

%(
typedef char *POINTER;
#define YYSTYPE POINTER

extern POINTER array( );
extern POINTER duplicate( );
extern POINTER implied_do_list( );
extern POINTER label( );
extern POINTER merge( );
extern POINTER type( );
%)

%%

program:
    optional_statement_list
    {
        summary( );
    }
    ;

optional_statement_list:
    /* NULL */
    |
        statement_list
    ;

statement_list:
    statement
    |
        statement_list statement
    ;

statement:

```



```

comment_statement
|
|label unlabeled_statement
|{
|    statement( $1 );
|}
|
;

comment_statement:
COMMENT
{
    if ( timer( $1 ) == 0 )
    {
        comment_statement( $1 );
    }
}
;

label:
LABEL
{
    $$ = label( $1 );
}
;

unlabeled_statement:
include_statement
|
program_statement
|
block_data_statement
|
function_statement
|
subroutine_statement
|
entry_statement
|
end_statement
|
specification_statement
|
executable_statement
|
format_statement
;

include_statement:
RW_INCLUDE character_constant
{
    include_statement( $2 );
}
;

program_statement:
RW_PROGRAM program_identifier
{
    program_statement( $2 );
}
;

program_identifier:
IDENTIFIER
{
    block( $1 );
    $$ = $1;
}
;

block_data_statement:
RW_BLOCK_DATA block_data_identifier
{
    block_data_statement( $2 );
}

```

```

;

block_data_identifier:
  IDENTIFIER
  {
    block( $1 );
    $$ = $1;
  }
;

function_statement:
  RW_FUNCTION function_identifier optional_formal_argument_list
  {
    function_statement( 0, $2, $3 );
  }
|
  type RW_FUNCTION function_identifier optional_formal_argument_list
  {
    function_statement( $1, $3, $4 );
  }
;

function_identifier:
  IDENTIFIER
  {
    block( $1 );
    $$ = $1;
  }
;

subroutine_statement:
  RW_SUBROUTINE subroutine_identifier
  {
    subroutine_statement( $2, 0 );
  }
|
  RW_SUBROUTINE subroutine_identifier optional_formal_argument_list
  {
    subroutine_statement( $2, $3 );
  }
;

subroutine_identifier:
  IDENTIFIER
  {
    block( $1 );
    $$ = $1;
  }
;

entry_statement:
  RW_ENTRY entry_identifier
  {
    entry_statement( $2, 0 );
  }
|
  RW_ENTRY entry_identifier optional_formal_argument_list
  {
    entry_statement( $2, $3 );
  }
;

entry_identifier:
  IDENTIFIER
  {
    $$ = $1;
  }
;

optional_formal_argument_list:
  '(' ' ' ')'
  {
    $$ = 0;
  }

```

```

    }
    |
    (' formal_argument_list ')
    {
        $$ = $2;
    }
    ;

formal_argument_list:
    formal_argument
    {
        $$ = merge( "%s", $1 );
    }
    |
    formal_argument_list ',' formal_argument
    {
        $$ = merge( "%s%s", $1, $3 );
    }
    ;

formal_argument:
    IDENTIFIER
    {
        $$ = $1;
    }
    |
    formal_argument_altername_return
    {
        $$ = $1;
    }
    ;

formal_argument_altername_return:
    '*'
    {
        $$ = duplicate( "*" );
    }
    ;

end_statement:
    RW_END
    {
        end_statement( );
    }
    ;

specification_statement:
    external_statement
    |
    intrinsic_statement
    |
    parameter_statement
    |
    dimension_statement
    |
    declaration_statement
    |
    save_statement
    |
    common_statement
    |
    equivalence_statement
    |
    implicit_statement
    |
    data_statement
    |
    namelist_statement
    ;

external_statement:
    RW_EXTERNAL external_list
    {
        external_statement( $2 );
    }

```

```

;

external_list:
    external
    {
        $$ = merge( "%s", $1 );
    }
    |
    external_list ',' external
    {
        $$ = merge( "%s%s", $1, $3 );
    }
;

external:
    IDENTIFIER
    {
        $$ = $1;
    }
;

intrinsic_statement:
    RW_INTRINSIC intrinsic_list
    {
        intrinsic_statement( $2 );
    }
;

intrinsic_list:
    intrinsic
    {
        $$ = merge( "%s", $1 );
    }
    |
    intrinsic_list ',' intrinsic
    {
        $$ = merge( "%s%s", $1, $3 );
    }
;

intrinsic:
    IDENTIFIER
    {
        $$ = $1;
    }
;

parameter_statement:
    RW_PARAMETER '(' parameter_list ')'
    {
        parameter_statement( $3 );
    }
;

parameter_list:
    parameter
    {
        $$ = merge( "%s", $1 );
    }
    |
    parameter_list ',' parameter
    {
        $$ = merge( "%s%s", $1, $3 );
    }
;

parameter:
    IDENTIFIER '=' expression
    {
        $$ = merge( "%s%s", $1, $3 );
    }
;

```

```

dimension_statement:
  RW_DIMENSION dimension_list
  {
    dimension_statement( $2 );
  }
;

dimension_list:
  dimension
  {
    $$ = merge( "{%s}", $1 );
  }
  |
  dimension_list ',' dimension
  {
    $$ = merge( "%s{%s}", $1, $3 );
  }
;

dimension:
  IDENTIFIER '(' subscript_list ')'
  {
    $$ = merge( "{%s}{%s}", $1, $3 );
  }
;

subscript_list:
  subscript
  {
    $$ = merge( "{%s}", $1 );
  }
  |
  subscript_list ',' subscript
  {
    $$ = merge( "%s{%s}", $1, $3 );
  }
;

subscript:
  upper_bound
  {
    $$ = $1;
  }
  |
  lower_bound ':' upper_bound
  {
    $$ = merge( "%s:%s", $1, $3 );
  }
;

lower_bound:
  expression
  {
    $$ = $1;
  }
;

upper_bound:
  lower_bound
  {
    $$ = $1;
  }
  |
  upper_bound_adjustable
  {
    $$ = $1;
  }
;

upper_bound_adjustable:
  '*'
  {
    $$ = duplicate( "*" );
  }
;

```

```

    ;
}

declaration_statement:
    type declaration_list
    {
        declaration_statement( $1, $2 );
    }
;

declaration_list:
    declaration
    {
        $$ = merge( "{%s}", $1 );
    }
    |
    declaration_list ',' declaration
    {
        $$ = merge( "%s{%s}", $1, $3 );
    }
;

declaration:
    IDENTIFIER
    {
        $$ = merge( "{%s}", $1 );
    }
    |
    IDENTIFIER '(' subscript_list ')'
    {
        $$ = merge( "%s{%s}", $1, $3 );
    }
;

type:
    type_name optional_type_length
    {
        $$ = type( $1, $2 );
    }
;

type_name:
    RW_CHARACTER
    {
        $$ = duplicate( "CHARACTER" );
    }
    |
    RW_COMPLEX
    {
        $$ = duplicate( "COMPLEX" );
    }
    |
    RW_DOUBLE_PRECISION
    {
        $$ = duplicate( "DOUBLE PRECISION" );
    }
    |
    RW_INTEGER
    {
        $$ = duplicate( "INTEGER" );
    }
    |
    RW_LOGICAL
    {
        $$ = duplicate( "LOGICAL" );
    }
    |
    RW_REAL
    {
        $$ = duplicate( "REAL" );
    }
    |
    RW_UNDEFINED
    {
        $$ = duplicate( "UNDEFINED" );
    }

```

```

;

optional_type_length:
/* NULL */
{
    $$ = 0;
}
|
type_length
{
    $$ = $1;
}
;

type_length:
/* INTEGER
{
    $$ = merge( "%s", $2 );
}
|
/* type_length_adjustable
{
    $$ = merge( "%s", $2 );
}
;

type_length_adjustable:
/* ( '(' , '*' , ')' )
{
    $$ = duplicate( "(" );
}
;

save_statement:
RW_SAVE optional_save_list
{
    save_statement( $2 );
}
;

optional_save_list:
/* NULL */
{
    $$ = 0;
}
|
save_list
{
    $$ = $1;
}
;

save_list:
save
{
    $$ = merge( "%s", $1 );
}
|
save_list ' ' save
{
    $$ = merge( "%s%s", $1, $3 );
}
;

save:
IDENTIFIER
{
    $$ = $1;
}
|
common_name
{
    $$ = $1;
}

```

```

;

common_statement:
  RW_COMMON optional_common_name common_list
  {
    common_statement( $2, $3 );
  }
;

optional_common_name:
  /* NULL */
  {
    $$ = 0;
  }
  |
  common_name
  {
    $$ = $1;
  }
;

common_name:
  '/' optional_identifier '/'
  {
    $$ = $2;
  }
;

optional_identifier:
  /* NULL */
  {
    $$ = 0;
  }
  |
  IDENTIFIER
  {
    $$ = $1;
  }
;

common_list:
  common
  {
    $$ = merge( "${s}", $1 );
  }
  |
  common_list ',' common
  {
    $$ = merge( "${s}${s}", $1, $3 );
  }
;

common:
  IDENTIFIER
  {
    $$ = merge( "${s}", $1 );
  }
  |
  IDENTIFIER '(' subscript_list ')'
  {
    $$ = merge( "${s}${s}", $1, $3 );
  }
;

equivalence_statement:
  RW_EQUIVALENCE equivalence_list
  {
    equivalence_statement( $2 );
  }
;

equivalence_list:
  equivalence

```



```

    {
        $$ = merge( "{%s}", $1 );
    }
    |
    equivalence_list ',' equivalence
    {
        $$ = merge( "%s{%s}", $1, $3 );
    }
    ;

equivalence:
    '(' variable_list ')'
    {
        $$ = $2;
    }
    ;

variable_list:
    variable
    {
        $$ = merge( "{%s}", $1 );
    }
    |
    variable_list ',' variable
    {
        $$ = merge( "%s{%s}", $1, $3 );
    }
    ;

implicit_statement:
    RW_IMPLICIT type '(' implicit_list ')'
    {
        implicit_statement( $2, $4 );
    }
    ;

implicit_list:
    implicit
    {
        $$ = merge( "{%s}", $1 );
    }
    |
    implicit_list ',' implicit
    {
        $$ = merge( "%s{%s}", $1, $3 );
    }
    ;

implicit:
    IDENTIFIER
    {
        $$ = $1;
    }
    |
    IDENTIFIER '-' IDENTIFIER
    {
        $$ = merge( "%s-%s", $1, $3 );
    }
    ;

namelist_statement:
    RW_NAMELIST namelist_name namelist_list
    {
        namelist_statement( $2, $3 );
    }
    ;

namelist_name:
    '/' IDENTIFIER '/'
    {
        $$ = $2;
    }
    ;

```

```

namelist_list:
    namelist
    {
        $$ = merge( "{%s}", $1 );
    }
    |
    namelist_list ',' namelist
    {
        $$ = merge( "%s{%s}", $1, $3 );
    }
    ;

namelist:
    IDENTIFIER
    {
        $$ = $1;
    }
    ;

data_statement:
    RW_DATA data_list
    {
        data_statement( $2 );
    }
    ;

data_list:
    data
    {
        $$ = merge( "{%s}", $1 );
    }
    |
    data_list optional_comma data
    {
        $$ = merge( "%s{%s}", $1, $3 );
    }
    ;

data:
    data_variable_list '/' data_constant_list '/'
    {
        $$ = merge( "{%s}{%s}", $1, $3 );
    }
    ;

data_variable_list:
    data_variable
    {
        $$ = merge( "{%s}", $1 );
    }
    |
    data_variable_list ',' data_variable
    {
        $$ = merge( "%s{%s}", $1, $3 );
    }
    ;

data_variable:
    variable
    {
        $$ = $1;
    }
    |
    data_implied_do_list
    {
        $$ = $1;
    }
    ;

data_implied_do_list:
    '(' data_variable_list ',' IDENTIFIER '=' expression_list ')'
    {
        $$ = implied_do_list( $2, $4, $6 );
    }

```

```

    }
;

data_constant_list:
    data_constant
    {
        $$ = merge( "{%s}", $1 );
    }
|
    data_constant_list ',' data_constant
    {
        $$ = merge( "%s{%s}", $1, $3 );
    }
;

data_constant:
    data_initialization
    {
        $$ = $1;
    }
|
    IDENTIFIER '*' data_initialization
    {
        $$ = merge( "%s * %s", $1, $3 );
    }
|
    INTEGER '*' data_initialization
    {
        $$ = merge( "%s * %s", $1, $3 );
    }
;

data_initialization:
    IDENTIFIER
    {
        $$ = $1;
    }
|
    character_constant
    {
        $$ = $1;
    }
|
    logical_constant
    {
        $$ = $1;
    }
|
    signed_numerical_constant
    {
        $$ = $1;
    }
;

signed_numerical_constant:
    numerical_constant
    {
        $$ = $1;
    }
|
    '+' numerical_constant %prec SIGN
    {
        $$ = merge( "+%s", $2 );
    }
|
    '-' numerical_constant %prec SIGN
    {
        $$ = merge( "-%s", $2 );
    }
;

expression:
    parenthesis_expression
    {
        $$ = $1;
    }

```

```

    simple_expression
    {
        $$ = $1;
    }
;

parenthesis_expression:
    '(' expression ')'
    {
        $$ = merge( "(%s)", $2 );
    }
;

simple_expression:
    variable
    {
        $$ = $1;
    }
    |
    constant
    {
        $$ = $1;
    }
    |
    arithmetic_expression
    {
        $$ = $1;
    }
    |
    character_expression
    {
        $$ = $1;
    }
    |
    relational_expression
    {
        $$ = $1;
    }
    |
    logical_expression
    {
        $$ = $1;
    }
    |
    unary_expression
    {
        $$ = $1;
    }
;

variable:
    IDENTIFIER
    {
        $$ = $1;
    }
    |
    IDENTIFIER string_subset
    {
        $$ = merge( "%s %s", $1, $2 );
    }
    |
    array
    {
        $$ = $1;
    }
;

array:
    IDENTIFIER '(' optional_expression_list ')'
    {
        $$ = array( $1, $3 );
    }
    |
    IDENTIFIER '(' optional_expression_list ')' string_subset
    {
        $$ = merge( "%s %s", array( $1, $3 ), $5 );
    }
;

```

```

    }
;

optional_expression_list:
/* NULL */
{
    $$ = 0;
}
|
expression_list
{
    $$ = $1;
}
;

expression_list:
expression
{
    $$ = merge( "%s)", $1 );
}
|
expression_list ',' expression
{
    $$ = merge( "%s%s)", $1, $3 );
}
;

string_subset:
(' optional_expression ':' optional_expression ')
{
    $$ = merge( "%s:%s)", $2, $4 );
}
;

optional_expression:
/* NULL */
{
    $$ = 0;
}
|
expression
{
    $$ = $1;
}
;

constant:
character_constant
{
    $$ = $1;
}
|
logical_constant
{
    $$ = $1;
}
|
numerical_constant
{
    $$ = $1;
}
;

character_constant:
HOLLERITH
{
    $$ = $1;
}
|
STRING
{
    $$ = $1;
}
;

```

```

logical_constant:
    RW_FALSE
    {
        $$ = duplicate( ".FALSE." );
    }
    |
    RW_TRUE
    {
        $$ = duplicate( ".TRUE." );
    }
    ;

numerical_constant:
    DOUBLE_PRECISION
    {
        $$ = $1;
    }
    |
    INTEGER
    {
        $$ = $1;
    }
    |
    REAL
    {
        $$ = $1;
    }
    ;

arithmetic_expression:
    expression '+' expression %prec '+'
    {
        $$ = merge( "%s + %s", $1, $3 );
    }
    |
    expression '-' expression %prec '-'
    {
        $$ = merge( "%s - %s", $1, $3 );
    }
    |
    expression '*' expression %prec '*'
    {
        $$ = merge( "%s * %s", $1, $3 );
    }
    |
    expression '/' expression %prec '/'
    {
        $$ = merge( "%s / %s", $1, $3 );
    }
    |
    expression EXPONENTIATE expression %prec EXPONENTIATE
    {
        $$ = merge( "%s ** %s", $1, $3 );
    }
    ;

character_expression:
    expression '/' '/' expression %prec CONCATENATE
    {
        $$ = merge( "%s // %s", $1, $4 );
    }
    ;

relational_expression:
    expression RW_EQ expression %prec RW_EQ
    {
        $$ = merge( "%s .EQ. %s", $1, $3 );
    }
    |
    expression RW_NE expression %prec RW_NE
    {
        $$ = merge( "%s .NE. %s", $1, $3 );
    }
    |
    expression RW_LT expression %prec RW_LT
    {

```

```

    $$ = merge( "%s .LT. %s", $1, $3 );
}
|
expression RW_LE expression %prec RW_LE
{
    $$ = merge( "%s .LE. %s", $1, $3 );
}
|
expression RW_GT expression %prec RW_GT
{
    $$ = merge( "%s .GT. %s", $1, $3 );
}
|
expression RW_GE expression %prec RW_GE
{
    $$ = merge( "%s .GE. %s", $1, $3 );
}
;

logical_expression:
    expression RW_AND expression %prec RW_AND
    {
        $$ = merge( "%s .AND. %s", $1, $3 );
    }
|
    expression RW_OR expression %prec RW_OR
    {
        $$ = merge( "%s .OR. %s", $1, $3 );
    }
|
    expression RW_EQV expression %prec RW_EQV
    {
        $$ = merge( "%s .EQV. %s", $1, $3 );
    }
|
    expression RW_NEQV expression %prec RW_NEQV
    {
        $$ = merge( "%s .NEQV. %s", $1, $3 );
    }
;

unary_expression:
    '+' expression %prec SIGN
    {
        $$ = merge( "+%s", $2 );
    }
|
    '-' expression %prec SIGN
    {
        $$ = merge( "-%s", $2 );
    }
|
    RW_NOT expression %prec RW_NOT
    {
        $$ = merge( ".NOT. %s", $2 );
    }
;

executable_statement:
    do_statement
|
    logical_if_statement
|
    block_if_statement
|
    else_statement
|
    else_if_statement
|
    end_if_statement
|
    subset_executable_statement
;

do_statement:
    RW_DO INTEGER IDENTIFIER '=' expression_list
    {

```

```

        do_statement( $2, $3, $5 );
    }
;

logical_if_statement:
    if_expression subset_executable_statement
    {
        logical_if_statement( );
    }
;

if_expression:
    RW_IF '(' expression ')'
    {
        if_expression( $3 );
    }
;

block_if_statement:
    RW_IF '(' expression ')' RW_THEN
    {
        block_if_statement( $3 );
    }
;

else_statement:
    RW_ELSE
    {
        else_statement( );
    }
;

else_if_statement:
    RW_ELSE_IF '(' expression ')' RW_THEN
    {
        else_if_statement( $3 );
    }
;

end_if_statement:
    RW_END_IF
    {
        end_if_statement( );
    }
;

subset_executable_statement:
    assignment_statement
    |
    assign_statement
    |
    arithmetic_if_statement
    |
    continue_statement
    |
    call_statement
    |
    return_statement
    |
    unconditional_go_to_statement
    |
    computed_go_to_statement
    |
    assigned_go_to_statement
    |
    stop_statement
    |
    pause_statement
    |
    io_statement
;

assignment_statement:

```



```

variable '=' expression
{
    assignment_statement( $1, $3 );
}
;

assign_statement:
    RW_ASSIGN INTEGER RW_TO IDENTIFIER
    {
        assign_statement( $2, $4 );
    }
;

arithmetic_if_statement:
    RW_IF '(' expression ')' integer_list
    {
        arithmetic_if_statement( $3, $5 );
    }
;

continue_statement:
    RW_CONTINUE
    {
        continue_statement( );
    }
;

call_statement:
    RW_CALL IDENTIFIER
    {
        call_statement( $2, 0 );
    }
    |
    RW_CALL IDENTIFIER optional_actual_argument_list
    {
        call_statement( $2, $3 );
    }
;

optional_actual_argument_list:
    '(' ')'
    {
        $$ = 0;
    }
    |
    '(' actual_argument_list ')'
    {
        $$ = $2;
    }
;

actual_argument_list:
    actual_argument
    {
        $$ = merge( "%s)", $1 );
    }
    |
    actual_argument_list ',' actual_argument
    {
        $$ = merge( "%s(%s)", $1, $3 );
    }
;

actual_argument:
    expression
    {
        $$ = $1;
    }
    |
    actual_argument_altername_return
    {
        $$ = $1;
    }
;

```

```

actual_argument_altername_return:
    '*' INTEGER
    {
        $$ = merge( "%s", $2 );
    }
    ;

return_statement:
    RW_RETURN optional_expression
    {
        return_statement( $2 );
    }
    ;

unconditional_go_to_statement:
    RW_GO_TO INTEGER
    {
        unconditional_go_to_statement( $2 );
    }
    ;

computed_go_to_statement:
    RW_GO_TO '(' integer_list ')' optional_comma expression
    {
        computed_go_to_statement( $3, $6 );
    }
    ;

assigned_go_to_statement:
    RW_GO_TO IDENTIFIER
    {
        assigned_go_to_statement( $2, 0 );
    }
    |
    RW_GO_TO IDENTIFIER optional_comma '(' integer_list ')'
    {
        assigned_go_to_statement( $2, $5 );
    }
    ;

optional_comma:
    /* NULL */
    |
    ','
    ;

integer_list:
    INTEGER
    {
        $$ = merge( "%s", $1 );
    }
    |
    integer_list ',' INTEGER
    {
        $$ = merge( "%s%s", $1, $3 );
    }
    ;

pause_statement:
    RW_PAUSE optional_expression
    {
        pause_statement( $2 );
    }
    ;

stop_statement:
    RW_STOP optional_expression
    {
        stop_statement( $2 );
    }
    ;

```

```

io_statement:
  open_statement
  |
  close_statement
  |
  inquire_statement
  |
  read_statement
  |
  write_statement
  |
  print_statement
  |
  backspace_statement
  |
  rewind_statement
  |
  endfile_statement
  ;

open_statement:
  RW_OPEN '(' control_information_list ')'
  {
    open_statement( $3 );
  }
  ;

close_statement:
  RW_CLOSE '(' control_information_list ')'
  {
    close_statement( $3 );
  }
  ;

inquire_statement:
  RW_INQUIRE '(' control_information_list ')'
  {
    inquire_statement( $3 );
  }
  ;

read_statement:
  RW_READ '(' control_information_list ')' optional_io_list
  {
    read_statement( $3, $5 );
  }
  |
  RW_READ control
  {
    read_statement( $2, 0 );
  }
  |
  RW_READ control ',' io_list
  {
    read_statement( $2, $4 );
  }
  ;

write_statement:
  RW_WRITE '(' control_information_list ')' optional_io_list
  {
    write_statement( $3, $5 );
  }
  ;

print_statement:
  RW_PRINT control
  {
    print_statement( $2, 0 );
  }
  |
  RW_PRINT control ',' io_list
  {

```

```

        print_statement( $2, $4 );
    }
;

backspace_statement:
    RW_BACKSPACE '(' control_information_list ')'
    {
        backspace_statement( $3 );
    }
|
    RW_BACKSPACE control
    {
        backspace_statement( $2 );
    }
;

rewind_statement:
    RW_REWIND '(' control_information_list ')'
    {
        rewind_statement( $3 );
    }
|
    RW_REWIND control
    {
        rewind_statement( $2 );
    }
;

endfile_statement:
    RW_ENDFILE '(' control_information_list ')'
    {
        endfile_statement( $3 );
    }
|
    RW_ENDFILE control
    {
        endfile_statement( $2 );
    }
;

control_information_list:
    control_information
    {
        $$ = merge( "%s", $1 );
    }
|
    control_information_list ',' control_information
    {
        $$ = merge( "%s%s", $1, $3 );
    }
;

control_information:
    control
    {
        $$ = $1;
    }
|
    IDENTIFIER '=' expression
    {
        $$ = merge( "%s = %s", $1, $3 );
    }
;

control:
    variable
    {
        $$ = $1;
    }
|
    constant
    {
        $$ = $1;
    }
;

```

```

    '*'
    {
        $$ = duplicate( "*" );
    }
;

optional_io_list:
/* NULL */
{
    $$ = 0;
}
|
io_list
{
    $$ = $1;
}
;

io_list:
io
{
    $$ = merge( "{%s}", $1 );
}
|
io_list ',' io
{
    $$ = merge( "%s{%s}", $1, $3 );
}
;

io:
expression
{
    $$ = $1;
}
|
io_implied_do_list
{
    $$ = $1;
}
;

io_implied_do_list:
'(' io_list ',' IDENTIFIER '=' expression_list ')'
{
    $$ = implied_do_list( $2, $4, $6 );
}
;

format_statement:
RW_FORMAT
{
    format_statement( $1 );
}
;

%%

FILE: ctimer/include/list.h

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#define LIST struct list_type
LIST
{
    char *identifier;
    char *block_name;
}

```

```

    int number;
    LIST *next;
};

extern LIST *end_list();
extern LIST *add_list();
extern int find_list();
extern void print_list();
extern void delete_list();

FILE: ctimer/library/Makefile

#
# Copyright 1991
# Georgia Institute of Technology
# Computer Engineering Research Laboratory
# Author: Stephen R. Wachtel
#

CC = cc -g
INCLUDE = ../include
CFLAGS = -I$(INCLUDE)
LIBRARY = library.a

OBJECTS = \
    alias.o \
    array.o \
    count.o \
    duplicate.o \
    hollerith.o \
    implied_do_list.o \
    label.o \
    link_list.o \
    list.o \
    lowercase.o \
    main.o \
    margin_printf.o \
    merge.o \
    non_blank.o \
    parse.o \
    print_level.o \
    stack.o \
    statement.o \
    summary.o \
    timer.o \
    type.o \
    yyerror.o \
    yygetc.o \
    yywrap.o

$(LIBRARY): $(OBJECTS)
    rm -f $(LIBRARY)
    ar crv $(LIBRARY) $(OBJECTS)
    ranlib $(LIBRARY)

.SUFFIXES: .c .o
.c.o:
    $(CC) -c $(CFLAGS) $<

clean:
    rm -f $(LIBRARY) $(OBJECTS)

FILE: ctimer/library/alias.c

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```

#include <stdio.h>
#include <string.h>

extern int yylineno;
extern char *duplicate( );
extern char *lowercase( );

#define ALIAS struct alias_type
ALIAS
{
    char *old_identifier;
    char *new_identifier;
};

static ALIAS alias_table[ ] =
{
    { "", "" }
};

#define ALIAS_TABLE ( sizeof( alias_table ) / sizeof( ALIAS ) )

char *alias( identifier )
register char *identifier;
{
    register int low, high;
    register int middle, test;

    lowercase( identifier );

    low = 0;
    high = ALIAS_TABLE - 1;

    while ( low <= high )
    {
        middle = ( low + high ) / 2;
        test = strcmp( identifier, alias_table[ middle ].old_identifier );

        if ( test < 0 )
        {
            high = middle - 1;
            continue;
        }

        if ( test > 0 )
        {
            low = middle + 1;
            continue;
        }

        fprintf( stderr, "line %d, %s aliased to %s\n", yylineno, identifier, alias_table[
middle ].new_identifier );
        return( alias_table[ middle ].new_identifier );
    }

    return( identifier );
} /* alias */

FILE: ctimer/library/array.c

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

extern char *list( );
extern char *merge( );

char *array( identifier, optional_expression_list )
register char *identifier;

```

```

register char *optional_expression_list;
{
    if ( optional_expression_list != (char *)0 )
        return( merge( "%s(%s)", identifier, list( optional_expression_list, ", " ) ) );
    else
        return( merge( "%s()", identifier ) );
} /* array */

```

FILE: ctimer/library/count.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```

int count( string, length, c )
register char *string;
register int length;
register char c;
{
    register int c_count = 0;

    while ( length != 0 )
    {
        if ( *string == c )
            c_count++;

        string++;
        length--;
    }

    return( c_count );
} /* count */

```

FILE: ctimer/library/duplicate.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```

#include <stdio.h>
#include <string.h>
#include <malloc.h>

```

```

char *duplicate( string )
register char *string;
{
    register char *temporary = (char *)NULL;

    if ( string != (char *)NULL )
    {
        if ( ( temporary = (char *)malloc( strlen( string ) + 1 ) ) != (char *)NULL )
            strcpy( temporary, string );
        else
            fprintf( stderr, "ERROR: duplicate( %s )\n", string );
    }

    return( temporary );
} /* duplicate */

```

FILE: ctimer/library/hollerith.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```



```

*/

#include <stdio.h>

char *hollerith( string, delimiter )
register char *string;
register char delimiter;
{
    int hollerith_length;
    register int string_length = 0;

    sscanf( string, "%dh", &hollerith_length );

    string[ string_length++ ] = delimiter;
    while ( hollerith_length != 0 )
    {
        if ( ( string[ string_length ] = yyinput( ) ) == '\n' )
        {
            yyunput( string[ string_length ] );
            break;
        }

        string_length++;
        hollerith_length--;
    }
    string[ string_length++ ] = delimiter;

    string[ string_length ] = '\0';
    return( string );
} /* hollerith */

FILE: ctimer/library/IMPLIED_DO_LIST.C

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

extern char *list( );
extern char *merge( );

char *IMPLIED_DO_LIST( variable_list, identifier, expression_list )
register char *variable_list;
register char *identifier;
register char *expression_list;
{
    return( merge( "(%s, %s = %s)", list( variable_list, ", " ), identifier, list(
expression_list, ", " ) ) );
} /* IMPLIED_DO_LIST */

FILE: ctimer/library/LABEL.C

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

extern int level;

char *label( string )
register char *string;
{
    if ( string != (char *)0 )
        margin_printf( "%d\t", atoi( string ) );
    else
        margin_printf( "\t" );
}

```

```

    while ( check_stack( string ) != 0 )
    {
        pull_stack( );
        level--;
    }

    return( string );
} /* label */

FILE: ctimer/library/link_list.c

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include <stdio.h>
#include <malloc.h>
#include "list.h"

LIST *end_list( list )
register LIST *list;
{
    if ( list != (LIST *)NULL )
    {
        while ( list->next != (LIST *)NULL )
            list = list->next;
    }

    return( list );
} /* end_list */

LIST *add_list( list, identifier, block_name, number )
register LIST **list;
register char *identifier;
register char *block_name;
register int number;
{
    register LIST *temporary = (LIST *)malloc( sizeof( LIST ) );

    temporary->identifier = identifier;
    temporary->block_name = block_name;
    temporary->number = number;
    temporary->next = (LIST *)NULL;

    if ( *list == (LIST *)NULL )
        *list = temporary;
    else
        end_list( *list )->next = temporary;

    return( temporary );
} /* add_list */

int find_list( list, identifier, block_name )
register LIST *list;
register char *identifier;
register char *block_name;
{
    while ( list != (LIST *)NULL )
    {
        if ( ( strcmp( identifier, list->identifier ) == 0 )
            && ( strcmp( block_name, list->block_name ) == 0 ) )
            return( list->number );

        list = list->next;
    }

    return( 0 );
} /* find_list */

void print_list( file, list )
register FILE *file;

```

```

register LIST *list;
{
    while ( list != (LIST *)NULL )
    {
        fprintf( file, "%d", list->number );

        if ( list->block_name == (char *)NULL )
            fprintf( file, " \"\"\" );
        else
            fprintf( file, " %s", list->block_name );

        if ( list->identifier == (char *)NULL )
            fprintf( file, " \"\"\" );
        else
            fprintf( file, " %s", list->identifier );

        fprintf( file, "\n" );

        list = list->next;
    }
} /* print_list */

void delete_list( list )
register LIST *list;
{
    if ( list != (LIST *)NULL )
    {
        delete_list( list->next );

        free( list );
    }
} /* delete_list */

FILE: ctimer/library/list.c

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

extern char *parse( );
extern char *merge( );

char *list( input_list, delimiter )
register char *input_list;
register char *delimiter;
{
    register char *output_list;
    register char *list;
    register char *temporary;

    output_list = parse( input_list );
    list = parse( input_list );

    while ( list != (char *)0 )
    {
        temporary = merge( "%s%s%s", output_list, delimiter, list );

        free( output_list );
        free( list );

        output_list = temporary;
        list = parse( input_list );
    }

    return( output_list );
} /* list */

FILE: ctimer/library/lowercase.c

/*
 * Copyright 1991

```

```

* Georgia Institute of Technology
* Computer Engineering Research Laboratory
* Author: Stephen R. Wachtel
*/

char *lowercase( string )
register char *string;
{
    register int index = 0;

    while ( string[ index ] != '\0' )
    {
        string[ index ] = tolower( string[ index ] );
        index++;
    }

    return( string );
} /* lowercase */

FILE: ctimer/library/main.c

/*
* Copyright 1991
* Georgia Institute of Technology
* Computer Engineering Research Laboratory
* Author: Stephen R. Wachtel
*/

#include <stdio.h>

extern FILE *yyin;
extern FILE *yyout;

#define PROGRAM argument[ 0 ]
#define INPUT_FILE argument[ 1 ]
#define OUTPUT_FILE argument[ 2 ]

int main( number_argument, argument )
int number_argument;
char *argument[ ];
{
    if ( number_argument == 1 )
    {
        yyin = stdin;
        yyout = stdout;

        yyparse( );
        exit( 0 );
    }

    if ( number_argument == 3 )
    {
        if ( ( yyin = fopen( INPUT_FILE, "r" ) ) == (FILE *)NULL )
        {
            fprintf( stderr, "%s: ERROR - unable to open input file '%s'\n", PROGRAM,
INPUT_FILE );
            exit( -1 );
        }

        if ( ( yyout = fopen( OUTPUT_FILE, "w" ) ) == (FILE *)NULL )
        {
            fprintf( stderr, "%s: ERROR - unable to open output file '%s'\n", PROGRAM,
OUTPUT_FILE );
            exit( -1 );
        }

        yyparse( );
        exit( 0 );
    }

    fprintf( stderr, "usage: %s <input file> <output file>\n", PROGRAM );
    exit( 0 );
} /* main */

```

FILE: ctimer/library/margin\_printf.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include <stdio.h>
#include <string.h>

extern FILE *yyin;
extern FILE *yyout;

static char buffer[ 256 * 20 ] = { 0 };

static void output_buffer( file )
register FILE *file;
{
#define LENGTH 72
    int length = LENGTH;
    int continuation = 0;
    int quote = 0;
    char temporary;

    while ( strlen( buffer ) > length )
    {
        if ( continuation++ != 0 )
            fprintf( file, "    &" );

        quote += count( buffer, length, '\\' );
        if ( ( quote % 2 ) == 0 )
        {
            while ( length != 0 )
            {
                if ( buffer[ length - 0 ] == '\\' )
                    break;

                if ( buffer[ length - 0 ] == '\\,' )
                    break;

                if ( buffer[ length - 1 ] == '\\' )
                    break;

                length--;
            }

            if ( length == 0 )
            {
                fprintf( stderr, "ERROR: margin_printf()\n" );
                exit( -1 );
            }
        }

        temporary = buffer[ length ];
        buffer[ length ] = '\0';
        fprintf( file, "%s\n", buffer );
        buffer[ length ] = temporary;

        strcpy( &buffer[ 0 ], &buffer[ length ] );
        length = LENGTH - 6;
    }

    if ( strlen( buffer ) != 0 )
    {
        if ( continuation++ != 0 )
            fprintf( file, "    &" );

        fprintf( file, "%s\n", buffer );
    }
} /* output_buffer */

void margin_printf( format, a, b, c, d, e )

```

```

char *format;
int a, b, c, d, e;
{
    char temporary[ 256 * 20 ];

    sprintf( temporary, format, a, b, c, d, e );
    strcat( buffer, temporary );

    if ( buffer[ strlen( buffer ) - 1 ] == '\n' )
    {
        buffer[ strlen( buffer ) - 1 ] = '\0';

        while ( buffer[ strlen( buffer ) - 1 ] == ' ' )
            buffer[ strlen( buffer ) - 1 ] = '\0';

        switch ( buffer[ 0 ] )
        {
            case '\0':
                fprintf( yyout, "\n" );
                break;

            case '*':
            case 'c':
            case 'C':
                fprintf( yyout, "%s\n", buffer );
                break;

            default:
                output_buffer( yyout );
        }

        buffer[ 0 ] = '\0';
    }
} /* margin_printf */

```

FILE: ctimer/library/merge.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include <stdio.h>
#include <string.h>
#include <malloc.h>

#define STRLEN( s ) ( strlen( s ) - 2 )

char *merge( string, a, b, c, d )
register char *string;
register char *a;
register char *b;
register char *c;
register char *d;
{
    register char *temporary = (char *)NULL;

    switch ( count( string, strlen( string ), '%' ) )
    {
        case 0:
            if ( ( temporary = (char *)malloc( strlen( string ) + 1 ) ) != (char *)NULL )
                sprintf( temporary, string );
            else
                fprintf( stderr, "ERROR: merge( %s )\n", string );
            break;

        case 1:
            if ( ( temporary = (char *)malloc( strlen( string ) + STRLEN( a ) + 1 ) ) !=
                (char *)NULL )
                sprintf( temporary, string, a );
            else
                fprintf( stderr, "ERROR: merge( %s, %s )\n", string, a );
            break;
    }
}

```

```

    case 2:
        if ( ( temporary = (char *)malloc( strlen( string ) + STRLEN( a ) + STRLEN( b ) + 1 ) ) != (char *)NULL )
            sprintf( temporary, string, a, b );
        else
            fprintf( stderr, "ERROR: merge( %s, %s, %s )\n", string, a, b );
            break;

    case 3:
        if ( ( temporary = (char *)malloc( strlen( string ) + STRLEN( a ) + STRLEN( b ) + STRLEN( c ) + 1 ) ) != (char *)NULL )
            sprintf( temporary, string, a, b, c );
        else
            fprintf( stderr, "ERROR: merge( %s, %s, %s, %s )\n", string, a, b, c );
            break;

    case 4:
        if ( ( temporary = (char *)malloc( strlen( string ) + STRLEN( a ) + STRLEN( b ) + STRLEN( c ) + STRLEN( d ) + 1 ) ) != (char *)NULL )
            sprintf( temporary, string, a, b, c, d );
        else
            fprintf( stderr, "ERROR: merge( %s, %s, %s, %s, %s )\n", string, a, b, c, d );
    };
    break;

    default:
        fprintf( stderr, "ERROR: merge( %s )\n", string );
        break;
}

return( temporary );
} /* merge */

```

FILE: ctimer/library/non\_blank.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```
#include <string.h>
```

```

char *non_blank( string )
register char *string;
{
    register int offset;
    register int length;

    length = strlen( string ) - 1;
    while ( ( string[ length ] == ' ' ) && ( string[ length ] != '\0' ) )
        string[ length-- ] = '\0';

    offset = 0;
    while ( ( string[ offset ] == ' ' ) && ( string[ offset ] != '\0' ) )
        string[ offset++ ] = '\0';

    strcpy( string, &string[ offset ] );

    if ( strlen( string ) != 0 )
        return( string );
    else
        return( 0 );
} /* non_blank */

```

FILE: ctimer/library/parse.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```

#include <string.h>

extern char *duplicate( );

char *parse( list )
register char *list;
{
    register int length = 0;
    register int brace = 0;
    register char *temporary = (char *)0;

    for (;;)
    {
        switch ( list[ length ] )
        {
            case '(':
                brace++;
                break;

            case ')':
                brace--;
                break;
        }

        if ( brace == 0 )
            break;

        length++;
    }

    if ( length != 0 )
    {
        list[ length ] = '\0';
        temporary = duplicate( list + 1 );
        strcpy( list, list + 1 + length );
    }
    else
    {
        if ( list[ length ] != '\0' )
        {
            temporary = duplicate( list );
            list[ length ] = '\0';
        }
    }

    return( temporary );
} /* parse */

FILE: ctimer/library/print_level.c

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

int level = 0;

void print_level( level )
register int level;
{
    if ( level != 0 )
    {
        while ( level-- != 0 )
            margin_p printf( " " );
    }
} /* print_level */

FILE: ctimer/library/stack.c

/*

```



```

* Copyright 1991
* Georgia Institute of Technology
* Computer Engineering Research Laboratory
* Author: Stephen R. Wachtel
*/

```

```
extern int level;
```

```
#define STACK 128
```

```
static struct
```

```
{
    char *label;
} stack[ STACK ];
static int pointer = 0;
```

```
int push_stack( label )
```

```
register char *label;
```

```
{
    if ( pointer != STACK )
    {
        stack[ pointer ].label = label;

        pointer++;
        return( 1 );
    }

    return( 0 );
} /* push_stack */
```

```
int check_stack( label )
```

```
register char *label;
```

```
{
    if ( pointer != 0 )
    {
        if ( strcmp( stack[ pointer - 1 ].label, label ) == 0 )
            return( 1 );
    }

    return( 0 );
} /* check_stack */
```

```
int pull_stack( )
```

```
{
    if ( pointer != 0 )
    {
        pointer--;

        free( stack[ pointer ].label );
        return( 1 );
    }

    return( 0 );
} /* pull_stack */
```

```
FILE: ctimer/library/statement.c
```

```

/*
* Copyright 1991
* Georgia Institute of Technology
* Computer Engineering Research Laboratory
* Author: Stephen R. Wachtel
*/

```

```
void statement( label )
```

```
register char *label;
```

```
{
} /* statement */
```

```
FILE: ctimer/library/summary.c
```

```
/*
```

```

* Copyright 1991
* Georgia Institute of Technology
* Computer Engineering Research Laboratory
* Author: Stephen R. Wachtel
*/

#include <stdio.h>
#include "list.h"

extern LIST *call_list;

void summary( )
{
    print_list( stdout, call_list );
} /* summary */

FILE: ctimer/library/timer.c

/*
* Copyright 1991
* Georgia Institute of Technology
* Computer Engineering Research Laboratory
* Author: Stephen R. Wachtel
*/

#include <stdio.h>
#include <string.h>
#include "list.h"

extern int level;

LIST *call_list = (LIST *)NULL;
char *block_name;
int timer_number = 0;

void block( identifier )
register char *identifier;
{
    block_name = identifier;
} /* block */

int timer( comment )
register char *comment;
{
    if ( strncmp( comment, "**LOOP*", 6 ) != 0 )
        return( 0 );

    if ( strcmp( comment, "**LOOP* PROLOGUE\n" ) == 0 )
    {
        label( 0 );
        print_level( level );
        margin_printf( "CALL timer_prologue()\n" );

        return( 1 );
    }

    if ( strcmp( comment, "**LOOP* START\n" ) == 0 )
        return( 1 );

    if ( strcmp( comment, "**LOOP* STOP\n" ) == 0 )
        return( 1 );

    if ( strcmp( comment, "**LOOP* EPILOGUE\n" ) == 0 )
    {
        label( 0 );
        print_level( level );
        margin_printf( "CALL timer_epilogue()\n" );

        return( 1 );
    }
}

```

```

    return( 0 );
} /* timer */

void start_timer( identifier )
register char *identifier;
{
    add_list( &call_list, identifier, block_name, ++timer_number );

    print_level( level );
    margin_printf( "CALL start_timer(%d)\n", timer_number );
    label( 0 );
} /* start_timer */

```

```

void stop_timer( identifier )
register char *identifier;
{
    label( 0 );
    print_level( level );
    margin_printf( "CALL stop_timer(%d)\n", timer_number );
} /* stop_timer */

```

FILE: ctimer/library/type.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

extern char *merge( );

char *type( type_name, optional_type_length )
register char *type_name;
register char *optional_type_length;
{
    if ( optional_type_length != (char *)0 )
        return( merge( "%s%s", type_name, optional_type_length ) );
    else
        return( type_name );
} /* type */

```

FILE: ctimer/library/yyerror.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include <stdio.h>

extern int yylineno;

void yyerror( string )
register char *string;
{
    fprintf( stderr, "line %d, %s\n", yylineno, string );
    exit( -1 );
} /* yyerror */

```

FILE: ctimer/library/yygetc.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology

```

```

* Computer Engineering Research Laboratory
* Author: Stephen R. Wachtel
*/

#include <stdio.h>
#include <ctype.h>

extern int yylineno;

int tab( length )
register int length;
{
    while ( length-- != 0 )
        yyunput( ' ' );

    return( ' ' );
} /* tab */

int yygetc( file )
register FILE *file;
{
    int c;
    int column[ 6 ];

loop:
    if ( ( c = getc( file ) ) == '\t' )
        c = tab( 6 );

    if ( c != '\n' )
        return( c );

    if ( ( column[ 0 ] = getc( file ) ) != ' ' )
        goto abort_0;
    if ( ( column[ 1 ] = getc( file ) ) != ' ' )
        goto abort_1;
    if ( ( column[ 2 ] = getc( file ) ) != ' ' )
        goto abort_2;
    if ( ( column[ 3 ] = getc( file ) ) != ' ' )
        goto abort_3;
    if ( ( column[ 4 ] = getc( file ) ) != ' ' )
        goto abort_4;
    if ( isspace( column[ 5 ] = getc( file ) ) )
        goto abort_5;

    yylineno++;
    goto loop;

abort_5:
    if ( column[ 5 ] == '\t' )
        tab( 1 );
    else
    {
        yyunput( column[ 5 ] );
        if ( column[ 5 ] == '\n' )
            yylineno++;
    }

abort_4:
    if ( column[ 4 ] == '\t' )
        tab( 2 );
    else
    {
        yyunput( column[ 4 ] );
        if ( column[ 4 ] == '\n' )
            yylineno++;
    }

abort_3:
    if ( column[ 3 ] == '\t' )
        tab( 3 );
    else
    {
        yyunput( column[ 3 ] );
        if ( column[ 3 ] == '\n' )
            yylineno++;
    }
}

```

```

abort_2:
    if ( column[ 2 ] == '\t' )
        tab( 4 );
    else
    {
        yyunput( column[ 2 ] );
        if ( column[ 2 ] == '\n' )
            yylineno++;
    }

abort_1:
    if ( column[ 1 ] == '\t' )
        tab( 5 );
    else
    {
        yyunput( column[ 1 ] );
        if ( column[ 1 ] == '\n' )
            yylineno++;
    }

abort_0:
    if ( column[ 0 ] == '\t' )
        tab( 6 );
    else
    {
        yyunput( column[ 0 ] );
        if ( column[ 0 ] == '\n' )
            yylineno++;
    }

    return( c );
} /* yygetc */

```

FILE: ctimer/library/yywrap.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

int yywrap( )
{
    return( 1 );
} /* yywrap */

```

FILE: ctimer/scanner.l

```

%{
/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */
%}

%a 10000
%e 10000
%k 10000
%n 10000
%o 10000
%p 10000

a  [aA]
b  [bB]
c  [cC]
d  [dD]
e  [eE]
f  [fF]
g  [gG]
h  [hH]
i  [iI]

```

```

j  [jJ]
k  [kK]
l  [lL]
m  [mM]
n  [nN]
o  [oO]
p  [pP]
q  [qQ]
r  [rR]
s  [sS]
t  [tT]
u  [uU]
v  [vV]
w  [wW]
x  [xX]
y  [yY]
z  [zZ]

%{
#include "grammar.h"
extern char *yylval;

#undef YYLMAX
#define YYLMAX (256*20)

extern char *duplicate( );
extern char *hollerith( );
extern char *non_blank( );
extern char *alias( );
%}

%%

^[\\*cC].*[\\n]  |
^[\\ ]*[\\n]  {
#ifdef DEBUG
    ECHO;
#endif
    yyval = duplicate( yytext );
    return( COMMENT );
}

[\\ ]  {
#ifdef DEBUG
    ECHO;
#endif
    /* return( '\\ ' ) */;
}

[\\&]  {
#ifdef DEBUG
    ECHO;
#endif
    return( '\\&' );
}

[\\(]  {
#ifdef DEBUG
    ECHO;
#endif
    return( '\\(' );
}

[\\)]  {
#ifdef DEBUG
    ECHO;
#endif
    return( '\\)' );
}

[\\*]  {

```

```

#ifdef DEBUG
    ECHO;
#endif
    return( '*' );
}

[\\*]\\* {
#ifdef DEBUG
    ECHO;
#endif
    return( EXPONENTIATE );
}

[\\+] {
#ifdef DEBUG
    ECHO;
#endif
    return( '+' );
}

[\\,] {
#ifdef DEBUG
    ECHO;
#endif
    return( ',' );
}

[\\-] {
#ifdef DEBUG
    ECHO;
#endif
    return( '-' );
}

[\\.] {
#ifdef DEBUG
    ECHO;
#endif
    return( '.' );
}

[\\/] {
#ifdef DEBUG
    ECHO;
#endif
    return( '/' );
}

[\\:] {
#ifdef DEBUG
    ECHO;
#endif
    return( ':' );
}

[\\=] {
#ifdef DEBUG
    ECHO;
#endif
    return( '=' );
}

[\\n] {
#ifdef DEBUG
    ECHO;
#endif
    /* return( '\\n' ) */;
}

[\\t] {
#ifdef DEBUG

```

```

    ECHO;
#endif
    /* return( '\t' ) */;
}

[\.]{a}{n}{d}[\.] {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_AND );
}

[\.]{e}{q}[\.] {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_EQ );
}

[\.]{e}{q}{v}[\.] {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_EQV );
}

[\.]{f}{a}{l}{s}{e}[\.] {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_FALSE );
}

[\.]{g}{e}[\.] {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_GE );
}

[\.]{g}{t}[\.] {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_GT );
}

[\.]{l}{e}[\.] {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_LE );
}

[\.]{l}{t}[\.] {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_LT );
}

[\.]{n}{e}[\.] {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_NE );
}

[\.]{n}{e}{q}{v}[\.] {
#ifdef DEBUG
    ECHO;

```



```

#endif
    return( RW_NEQV );
}

[\.]{n}{o}{t}{\.} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_NOT );
}

[\.]{o}{r}{\.} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_OR );
}

[\.]{t}{r}{u}{e}{\.} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_TRUE );
}

{a}{s}{s}{i}{g}{n} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_ASSIGN );
}

{b}{a}{c}{k}{s}{p}{a}{c}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_BACKSPACE );
}

{b}{l}{o}{c}{k}{\ }*(d){a}{t}{a} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_BLOCK_DATA );
}

{c}{a}{l}{l} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_CALL );
}

{c}{h}{a}{r}{a}{c}{t}{e}{r} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_CHARACTER );
}

{c}{l}{o}{s}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_CLOSE );
}

{c}{o}{m}{m}{o}{n} {
#ifdef DEBUG
    ECHO;
#endif

```

```

        return( RW_COMMON );
    }

{c}{o}{m}{p}{l}{e}{x} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_COMPLEX );
}

{c}{o}{n}{t}{i}{n}{u}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_CONTINUE );
}

{d}{a}{t}{a} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_DATA );
}

{d}{i}{m}{e}{n}{s}{i}{o}{n} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_DIMENSION );
}

{d}{o} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_DO );
}

{d}{o}{u}{b}{l}{e}{\ }*{p}{r}{e}{c}{i}{s}{i}{o}{n} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_DOUBLE_PRECISION );
}

{e}{l}{s}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_ELSE );
}

{e}{l}{s}{e}{\ }*{i}{f} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_ELSE_IF );
}

{e}{n}{d} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_END );
}

{e}{n}{d}{\ }*{i}{f} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_END_IF );
}

```

```

    }

    {e}{n}{d}{f}{i}{l}{e} {
#ifdef DEBUG
        ECHO;
#endif
        return( RW_ENDFILE );
    }

    {e}{n}{t}{r}{y} {
#ifdef DEBUG
        ECHO;
#endif
        return( RW_ENTRY );
    }

    {e}{q}{u}{i}{v}{a}{l}{e}{n}{c}{e} {
#ifdef DEBUG
        ECHO;
#endif
        return( RW_EQUIVALENCE );
    }

    {e}{x}{t}{e}{r}{n}{a}{l} {
#ifdef DEBUG
        ECHO;
#endif
        return( RW_EXTERNAL );
    }

    {f}{o}{r}{m}{a}{t}.* {
#ifdef DEBUG
        ECHO;
#endif
        yyval = duplicate( yytext );
        return( RW_FORMAT );
    }

    {f}{u}{n}{c}{t}{i}{o}{n} {
#ifdef DEBUG
        ECHO;
#endif
        return( RW_FUNCTION );
    }

    {g}{o}{[ \ ]}*{t}{o} {
#ifdef DEBUG
        ECHO;
#endif
        return( RW_GO_TO );
    }

    {i}{f} {
#ifdef DEBUG
        ECHO;
#endif
        return( RW_IF );
    }

    {i}{m}{p}{l}{i}{c}{i}{t} {
#ifdef DEBUG
        ECHO;
#endif
        return( RW_IMPLICIT );
    }

    {i}{n}{c}{l}{u}{d}{e} {
#ifdef DEBUG
        ECHO;
#endif
        return( RW_INCLUDE );
    }

```

```

    }

    {i}{n}{q}{u}{i}{r}{e} {
#ifdef DEBUG
        ECHO;
#endif
        return( RW_INQUIRE );
    }

    {i}{n}{t}{e}{g}{e}{r} {
#ifdef DEBUG
        ECHO;
#endif
        return( RW_INTEGER );
    }

    {i}{n}{t}{r}{i}{n}{s}{i}{c} {
#ifdef DEBUG
        ECHO;
#endif
        return( RW_INTRINSIC );
    }

    {l}{o}{g}{i}{c}{a}{l} {
#ifdef DEBUG
        ECHO;
#endif
        return( RW_LOGICAL );
    }

    {n}{a}{m}{e}{l}{i}{s}{t} {
#ifdef DEBUG
        ECHO;
#endif
        return( RW_NAMELIST );
    }

    {o}{p}{e}{n} {
#ifdef DEBUG
        ECHO;
#endif
        return( RW_OPEN );
    }

    {p}{a}{r}{a}{m}{e}{t}{e}{r} {
#ifdef DEBUG
        ECHO;
#endif
        return( RW_PARAMETER );
    }

    {p}{a}{u}{s}{e} {
#ifdef DEBUG
        ECHO;
#endif
        return( RW_PAUSE );
    }

    {p}{r}{i}{n}{t} {
#ifdef DEBUG
        ECHO;
#endif
        return( RW_PRINT );
    }

    {p}{r}{o}{g}{r}{a}{m} {
#ifdef DEBUG
        ECHO;
#endif
        return( RW_PROGRAM );
    }

```

```
{r}{e}{a}{d} {  
#ifdef DEBUG  
    ECHO;  
#endif  
    return( RW_READ );  
}
```

```
{r}{e}{a}{l} {  
#ifdef DEBUG  
    ECHO;  
#endif  
    return( RW_REAL );  
}
```

```
{r}{e}{t}{u}{r}{n} {  
#ifdef DEBUG  
    ECHO;  
#endif  
    return( RW_RETURN );  
}
```

```
{r}{e}{w}{i}{n}{d} {  
#ifdef DEBUG  
    ECHO;  
#endif  
    return( RW_REWIND );  
}
```

```
{s}{a}{v}{e} {  
#ifdef DEBUG  
    ECHO;  
#endif  
    return( RW_SAVE );  
}
```

```
{s}{t}{o}{p} {  
#ifdef DEBUG  
    ECHO;  
#endif  
    return( RW_STOP );  
}
```

```
{s}{u}{b}{r}{o}{u}{t}{i}{n}{e} {  
#ifdef DEBUG  
    ECHO;  
#endif  
    return( RW_SUBROUTINE );  
}
```

```
{t}{h}{e}{n} {  
#ifdef DEBUG  
    ECHO;  
#endif  
    return( RW_THEN );  
}
```

```
{t}{o} {  
#ifdef DEBUG  
    ECHO;  
#endif  
    return( RW_TO );  
}
```

```
{w}{r}{i}{t}{e} {  
#ifdef DEBUG  
    ECHO;  
#endif  
    return( RW_WRITE );  
}
```

```

{u}{n}{d}{e}{f}{i}{n}{e}{d} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_UNDEFINED );
}

[%a-zA-Z][_a-zA-Z0-9]* {
#ifdef DEBUG
    ECHO;
#endif
    yylval = duplicate( alias( yytext ) );
    return( IDENTIFIER );
}

^[0-9 ][0-9 ][0-9 ][0-9 ][0-9 ][\ ] {
#ifdef DEBUG
    ECHO;
#endif
    yylval = duplicate( non_blank( yytext ) );
    return( LABEL );
}

[0-9]+ |
[0-9]+/\.[a-zA-Z]+\.\. {
#ifdef DEBUG
    ECHO;
#endif
    yylval = duplicate( yytext );
    return( INTEGER );
}

[0-9]+\.[0-9]*([eE][\+\-]?[0-9]+)? |
[0-9]*\.[0-9]+([eE][\+\-]?[0-9]+)? |
[0-9]+([eE][\+\-]?[0-9]+)? {
#ifdef DEBUG
    ECHO;
#endif
    yylval = duplicate( yytext );
    return( REAL );
}

[0-9]+\.[0-9]*([dD][\+\-]?[0-9]+)? |
[0-9]*\.[0-9]+([dD][\+\-]?[0-9]+)? |
[0-9]+([dD][\+\-]?[0-9]+)? {
#ifdef DEBUG
    ECHO;
#endif
    yylval = duplicate( yytext );
    return( DOUBLE_PRECISION );
}

\[^\']*\\' |
\[^\"]*\\\" {
#ifdef DEBUG
    ECHO;
#endif
    yytext[ 0 ] = '\\';
    yytext[ strlen( yytext ) - 1 ] = '\\';
    yylval = duplicate( yytext );
    return( STRING );
}

[0-9]+[hH] {
#ifdef DEBUG
    ECHO;
#endif
    yylval = duplicate( hollerith( yytext, '\\' ) );
    return( HOLLERITH );
}

```

FILE: ctimer/statement/Makefile

```
#
# Copyright 1991
# Georgia Institute of Technology
# Computer Engineering Research Laboratory
# Author: Stephen R. Wachtel
#
```

```
CC = cc -g
INCLUDE = ../include
CFLAGS = -I$(INCLUDE)
LIBRARY = statement.a
```

```
OBJECTS = \
    arithmetic_if_statement.o \
    assign_statement.o \
    assigned_go_to_statement.o \
    assignment_statement.o \
    backspace_statement.o \
    block_data_statement.o \
    block_if_statement.o \
    call_statement.o \
    close_statement.o \
    comment_statement.o \
    common_statement.o \
    computed_go_to_statement.o \
    continue_statement.o \
    data_statement.o \
    declaration_statement.o \
    dimension_statement.o \
    do_statement.o \
    else_if_statement.o \
    else_statement.o \
    end_if_statement.o \
    end_statement.o \
    endfile_statement.o \
    entry_statement.o \
    equivalence_statement.o \
    external_statement.o \
    format_statement.o \
    function_statement.o \
    implicit_statement.o \
    include_statement.o \
    inquire_statement.o \
    intrinsic_statement.o \
    logical_if_statement.o \
    namelist_statement.o \
    open_statement.o \
    parameter_statement.o \
    pause_statement.o \
    print_statement.o \
    program_statement.o \
    read_statement.o \
    return_statement.o \
    rewind_statement.o \
    save_statement.o \
    stop_statement.o \
    subroutine_statement.o \
    unconditional_go_to_statement.o \
    write_statement.o
```

```
$(LIBRARY): $(OBJECTS)
    rm -f $(LIBRARY)
    ar crv $(LIBRARY) $(OBJECTS)
    ranlib $(LIBRARY)
```

```
.SUFFIXES: .c .o
.c.o:
    $(CC) -c $(CFLAGS) $<
```

```
clean:
    rm -f $(LIBRARY) $(OBJECTS)
```

```
FILE: ctimer/statement/arithmetic_if_statement.c
```

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```

extern int level;
extern char *list( );

```

```

void arithmetic_if_statement( expression, label_list )
register char *expression;
register char *label_list;
{
    print_level( level );
    margin_printf( "IF (%s) %s\n", expression, list( label_list, ", " ) );
} /* arithmetic_if_statement */

```

FILE: ctimer/statement/assign\_statement.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```

extern int level;

```

```

void assign_statement( label, identifier )
register char *label;
register char *identifier;
{
    print_level( level );
    margin_printf( "ASSIGN %s TO %s\n", label, identifier );
} /* assign_statement */

```

FILE: ctimer/statement/assigned go\_to\_statement.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```

extern int level;
extern char *list( );

```

```

void assigned_go_to_statement( identifier, optional_label_list )
register char *identifier;
register char *optional_label_list;
{
    if ( optional_label_list != 0 )
    {
        print_level( level );
        margin_printf( "GO TO %s, (%s)\n", identifier, list( optional_label_list, ", " ) );
    }
    else
    {
        print_level( level );
        margin_printf( "GO TO %s\n", identifier );
    }
} /* assigned_go_to_statement */

```

FILE: ctimer/statement/assignment\_statement.c



```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```
extern int level;
```

```

void assignment_statement( variable, expression )
register char *variable;
register char *expression;
{
    print_level( level );
    margin_printf( "%s = %s\n", variable, expression );
} /* assignment_statement */

```

```
FILE: ctimer/statement/backspace_statement.c
```

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```
extern int level;
extern char *list( );
```

```

void backspace_statement( control_list )
register char *control_list;
{
    print_level( level );
    margin_printf( "BACKSPACE (%s)\n", list( control_list, " " ) );
} /* backspace_statement */

```

```
FILE: ctimer/statement/block_data_statement.c
```

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```
extern int level;
```

```

void block_data_statement( identifier )
register char *identifier;
{
    print_level( level );
    margin_printf( "BLOCK DATA %s\n", identifier );
} /* block_data_statement */

```

```
FILE: ctimer/statement/block_if_statement.c
```

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```
extern int level;
```

```
void block_if_statement( expression )
```

```

register char *expression;
{
    print_level( level );
    margin_printf( "IF (%s) THEN\n", expression );

    level++;
} /* block_if_statement */

```

FILE: ctimer/statement/call\_statement.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

extern int level;
extern char *list( );

void call_statement( identifier, optional_actual_argument_list )
register char *identifier;
register char *optional_actual_argument_list;
{
    start_timer( identifier );

    if ( optional_actual_argument_list != 0 )
    {
        print_level( level );
        margin_printf( "CALL %s(%s)\n", identifier, list( optional_actual_argument_list,
", " ) );
    }
    else
    {
        print_level( level );
        margin_printf( "CALL %s()\n", identifier );
    }

    stop_timer( identifier );
} /* call_statement */

```

FILE: ctimer/statement/close\_statement.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

extern int level;
extern char *list( );

void close_statement( control_list )
register char *control_list;
{
    print_level( level );
    margin_printf( "CLOSE (%s)\n", list( control_list, " " ) );
} /* close_statement */

```

FILE: ctimer/statement/comment\_statement.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

void comment_statement( string )

```

```

register char *string;
{
    margin_printf( "%s", string );
} /* comment_statement */

```

FILE: ctimer/statement/common\_statement.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

extern int level;
extern char *parse( );
extern char *list( );

void common_statement( optional_common_name, common_list )
register char *optional_common_name;
register char *common_list;
{
    register char *common;
    register char *identifier;
    register char *optional_subscript_list;

    print_level( level );
    margin_printf( "COMMON " );

    if ( optional_common_name != 0 )
        margin_printf( "%s/ ", optional_common_name );

    while ( common = parse( common_list ) )
    {
        identifier = parse( common );
        optional_subscript_list = parse( common );

        margin_printf( "%s", identifier );
        if ( optional_subscript_list != 0 )
            margin_printf( "({s)", list( optional_subscript_list, ", " ) );

        if ( strlen( common_list ) != 0 )
            margin_printf( ", " );
    }

    margin_printf( "\n" );
} /* common_statement */

```

FILE: ctimer/statement/computed\_go\_to\_statement.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

extern int level;
extern char *list( );

void computed_go_to_statement( label_list, expression )
register char *label_list;
register char *expression;
{
    print_level( level );
    margin_printf( "GO TO (%s), %s\n", list( label_list, ", " ), expression );
} /* computed_go_to_statement */

```

FILE: ctimer/statement/continue\_statement.c

```

/*

```

```

/* Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```
extern int level;
```

```

void continue_statement( )
{
    print_level( level );
    margin_printf( "CONTINUE\n" );
} /* continue_statement */

```

```
FILE: ctimer/statement/data_statement.c
```

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```

extern int level;
extern char *parse( );
extern char *list( );

```

```

void data_statement( data_list )
register char *data_list;
{
    register char *data;
    register char *variable_list;
    register char *constant_list;

    print_level( level );
    margin_printf( "DATA " );

    while ( data = parse( data_list ) )
    {
        variable_list = parse( data );
        constant_list = parse( data );

        margin_printf( "%s /%s/", list( variable_list, " " ), list( constant_list, " " ) );

        if ( strlen( data_list ) != 0 )
            margin_printf( ", " );

        margin_printf( "\n" );
    } /* data_statement */
}

```

```
FILE: ctimer/statement/declaration_statement.c
```

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```

extern int level;
extern char *parse( );
extern char *list( );

```

```

void declaration_statement( type, declaration_list )
register char *type;
register char *declaration_list;
{
    register char *declaration;
    register char *identifier;
}

```

```

register char *optional_subscript_list;

print_level( level );
margin_printf( "%s ", type );

while ( declaration = parse( declaration_list ) )
{
    identifier = parse( declaration );
    optional_subscript_list = parse( declaration );

    margin_printf( "%s", identifier );
    if ( optional_subscript_list != 0 )
        margin_printf( "(%s)", list( optional_subscript_list, ", " ) );

    if ( strlen( declaration_list ) != 0 )
        margin_printf( ", " );
}

margin_printf( "\n" );
} /* declaration_statement */

```

FILE: ctimer/statement/dimension\_statement.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

extern int level;
extern char *parse( );
extern char *list( );

void dimension_statement( dimension_list )
register char *dimension_list;
{
    register char *dimension;
    register char *identifier;
    register char *subscript_list;

    print_level( level );
    margin_printf( "DIMENSION " );

    while ( dimension = parse( dimension_list ) )
    {
        identifier = parse( dimension );
        subscript_list = parse( dimension );

        margin_printf( "%s(%s)", identifier, list( subscript_list, ", " ) );

        if ( strlen( dimension_list ) != 0 )
            margin_printf( ", " );
    }

    margin_printf( "\n" );
} /* dimension_statement */

```

FILE: ctimer/statement/do\_statement.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

extern int level;
extern char *list( );

void do_statement( label, identifier, expression_list )
register char *label;
register char *identifier;

```

```

register char *expression_list;
{
    push_stack( label );

    print_level( level );
    margin_printf( "DO %s %s = %s\n", label, identifier, list( expression_list, ", " ) );

    level++;
} /* do_statement */

```

FILE: ctimer/statement/else\_if\_statement.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```
extern int level;
```

```

void else_if_statement( expression )
register char *expression;
{
    level--;

    print_level( level );
    margin_printf( "ELSE IF (%s) THEN\n", expression );

    level++;
} /* else_if_statement */

```

FILE: ctimer/statement/else\_statement.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```
extern int level;
```

```

void else_statement( )
{
    level- ;

    print_level( level );
    margin_printf( "ELSE\n" );

    level++;
} /* else_statement */

```

FILE: ctimer/statement/end\_if\_statement.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```
extern int level;
```

```

void end_if_statement( )
{
    level--;

    print_level( level );
}

```

```

    margin_printf( "END IF\n" );
} /* end_if_statement */

```

FILE: ctimer/statement/end\_statement.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```
extern int level;
```

```

void end_statement( )
{
    print_level( level );
    margin_printf( "END\n" );
} /* end_statement */

```

FILE: ctimer/statement/endfile\_statement.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```
extern int level;
extern char *list( );
```

```

void endfile_statement( control_list )
register char *control_list;
{
    print_level( level );
    margin_printf( "ENDFILE (%s)\n", list( control_list, " " ) );
} /* endfile_statement */

```

FILE: ctimer/statement/entry\_statement.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```
extern int level;
extern char *list( );
```

```

void entry_statement( identifier, optional_formal_argument_list )
register char *identifier;
register char *optional_formal_argument_list;
{
    if ( optional_formal_argument_list != 0 )
    {
        print_level( level );
        margin_printf( "ENTRY %s(%s)\n", identifier, list( optional_formal_argument_list,
        " " ) );
    }
    else
    {
        print_level( level );
        margin_printf( "ENTRY %s()\n", identifier );
    }
} /* entry_statement */

```

FILE: ctimer/statement/equivalence\_statement.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

extern int level;
extern char *parse();
extern char *list();

void equivalence_statement( equivalence_list )
register char *equivalence_list;
{
    register char *variable_list;

    print_level( level );
    margin_printf( "EQUIVALENCE " );

    while ( variable_list = parse( equivalence_list ) )
    {
        margin_printf( "(%s)", list( variable_list, ", " ) );

        if ( strlen( equivalence_list ) != 0 )
            margin_printf( ", " );
    }

    margin_printf( "\n" );
} /* equivalence_statement */

```

FILE: ctimer/statement/external\_statement.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

extern int level;
extern char *list();

void external_statement( external_list )
register char *external_list;
{
    print_level( level );
    margin_printf( "EXTERNAL %s\n", list( external_list, ", " ) );
} /* external_statement */

```

FILE: ctimer/statement/format\_statement.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

extern int level;

void format_statement( format )
register char *format;
{
    format[0] = 'F';
    format[1] = 'O';
    format[2] = 'R';
    format[3] = 'M';
    format[4] = 'A';
}

```



```

    format(5) = 'I';

    print_level( level );
    margin_printf( "%s\n", format );
} /* format_statement */

FILE: ctimer/statement/function_statement.c

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

extern int level;
extern char *list();

void function_statement( optional_type, identifier, optional_formal_argument_list )
register char *optional_type;
register char *identifier;
register char *optional_formal_argument_list;
{
    print_level( level );
    if ( optional_type != 0 )
        margin_printf( "%s ", optional_type );

    if ( optional_formal_argument_list != 0 )
        margin_printf( "FUNCTION %s(%s)\n", identifier, list(
optional_formal_argument_list, " " ) );
    else
        margin_printf( "FUNCTION %s()\n", identifier );
} /* function_statement */

```

FILE: ctimer/statement/implicit\_statement.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

extern int level;
extern char *list();

void implicit_statement( type, implicit_list )
register char *type;
register char *implicit_list;
{
    print_level( level );
    margin_printf( "IMPLICIT %s(%s)\n", type, list( implicit_list, " " ) );
} /* implicit_statement */

```

FILE: ctimer/statement/include\_statement.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

extern int level;

void include_statement( filename )
register char *filename;
{
    print_level( level );
}

```

```
margin_printf( "INCLUDE %s\n", filename );
} /* include_statement */
```

FILE: ctimer/statement/inquire\_statement.c

```
/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

extern int level;

void inquire_statement( control_list )
register char *control_list;
{
    print_level( level );
    margin_printf( "INQUIRE (%s)\n", list( control_list, " " ) );
} /* inquire_statement */
```

FILE: ctimer/statement/intrinsic\_statement.c

```
/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

extern int level;
extern char *list( );

void intrinsic_statement( intrinsic_list )
register char *intrinsic_list;
{
    print_level( level );
    margin_printf( "INTRINSIC %s\n", list( intrinsic_list, " " ) );
} /* intrinsic_statement */
```

FILE: ctimer/statement/logical\_if\_statement.c

```
/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

extern int level;

void logical_if_statement( )
{
    level--;
    label( 0 );

    print_level( level );
    margin_printf( "END IF\n" );
} /* logical_if_statement */

void if_expression( expression )
register char *expression;
{
    print_level( level );
    margin_printf( "IF (%s) THEN\n", expression );

    level+ ;
    label( 0 );
}
```

```

} /* if_expression */

```

```

FILE: ctimer/statement/namelist_statement.c

```

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

extern int level;
extern char *list( );

void namelist_statement( namelist_name, namelist_list )
register char *namelist_name;
register char *namelist_list;
{
    print_level( level );
    margin_printf( "NAMELIST %s/ %s\n", namelist_name, list( namelist_list, ", " ) );
} /* namelist_statement */

```

```

FILE: ctimer/statement/open_statement.c

```

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

extern int level;
extern char *list( );

void open_statement( control_list )
register char *control_list;
{
    print_level( level );
    margin_printf( "OPEN (%s)\n", list( control_list, ", " ) );
} /* open_statement */

```

```

FILE: ctimer/statement/parameter_statement.c

```

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

extern int level;
extern char *parse( );

void parameter_statement( parameter_list )
register char *parameter_list;
{
    register char *parameter;
    register char *identifier;
    register char *expression;

    print_level( level );
    margin_printf( "PARAMETER (" );

    while ( parameter = parse( parameter_list ) )
    {
        identifier = parse( parameter );
        expression = parse( parameter );

        margin_printf( "%s = %s", identifier, expression );
    }
}

```

```

        if ( strlen( parameter_list ) != 0 )
            margin_printf( " ", " );
    }

    margin_printf( ")\n" );
} /* parameter_statement */

```

FILE: ctimer/statement/pause\_statement.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

extern int level;

void pause_statement( optional_expression )
register char *optional_expression;
{
    if ( optional_expression != 0 )
    {
        print_level( level );
        margin_printf( "PAUSE %s\n", optional_expression );
    }
    else
    {
        print_level( level );
        margin_printf( "PAUSE\n" );
    }
} /* pause_statement */

```

FILE: ctimer/statement/print\_statement.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

extern int level;
extern char *list( );

void print_statement( control_list, optional_io_list )
register char *control_list;
register char *optional_io_list;
{
    if ( optional_io_list != 0 )
    {
        print_level( level );
        margin_printf( "PRINT (%s) %s\n", list( control_list, " ", " ), list(
optional_io_list, " ", " ) );
    }
    else
    {
        print_level( level );
        margin_printf( "PRINT (%s)\n", list( control_list, " ", " ) );
    }
} /* print_statement */

```

FILE: ctimer/statement/program\_statement.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```
extern int level;
```

```
void program_statement( identifier )
register char *identifier;
{
    print_level( level );
    margin_printf( "PROGRAM %s\n", identifier );
} /* program_statement */
```

```
FILE: ctimer/statement/read_statement.c
```

```
/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */
```

```
extern int level;
extern char *list( );
```

```
void read_statement( control_list, optional_io_list )
register char *control_list;
register char *optional_io_list;
{
    if( optional_io_list != 0 )
    {
        print_level( level );
        margin_printf( "READ (%s) %s\n", list( control_list, " " ), list(
optional_io_list, " " ) );
    }
    else
    {
        print_level( level );
        margin_printf( "READ (%s)\n", list( control_list, " " ) );
    }
} /* read_statement */
```

```
FILE: ctimer/statement/return_statement.c
```

```
/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */
```

```
extern int level;
```

```
void return_statement( expression )
register char *expression;
{
    if ( expression != 0 )
    {
        print_level( level );
        margin_printf( "RETURN %s\n", expression );
    }
    else
    {
        print_level( level );
        margin_printf( "RETURN\n" );
    }
} /* return_statement */
```

```
FILE: ctimer/statement/rewind_statement.c
```

```
/*
 * Copyright 1991
```

```

* Georgia Institute of Technology
* Computer Engineering Research Laboratory
* Author: Stephen R. Wachtel
*/

extern int level;
extern char *list( );

void rewind_statement( control_list )
register char *control_list;
{
    print_level( level );
    margin_printf( "REWIND (%s)\n", list( control_list, " " ) );
} /* rewind_statement */

```

FILE: ctimer/statement/save\_statement.c

```

/*
* Copyright 1991
* Georgia Institute of Technology
* Computer Engineering Research Laboratory
* Author: Stephen R. Wachtel
*/

extern int level;
extern char *list( );

void save_statement( save_list )
register char *save_list;
{
    print_level( level );
    margin_printf( "SAVE %s\n", list( save_list, " " ) );
} /* save_statement */

```

FILE: ctimer/statement/stop\_statement.c

```

/*
* Copyright 1991
* Georgia Institute of Technology
* Computer Engineering Research Laboratory
* Author: Stephen R. Wachtel
*/

extern int level;

void stop_statement( optional_expression )
register char *optional_expression;
{
    if ( optional_expression != 0 )
    {
        print_level( level );
        margin_printf( "STOP %s\n", optional_expression );
    }
    else
    {
        print_level( level );
        margin_printf( "STOP\n" );
    }
} /* stop_statement */

```

FILE: ctimer/statement/subroutine\_statement.c

```

/*
* Copyright 1991
* Georgia Institute of Technology
* Computer Engineering Research Laboratory
* Author: Stephen R. Wachtel
*/

```

```
extern int level;
extern char *list();
```

```
void subroutine_statement( identifier, optional_formal_argument_list )
register char *identifier;
register char *optional_formal_argument_list;
{
    if ( optional_formal_argument_list != 0 )
    {
        print_level( level );
        margin_printf( "SUBROUTINE %s(%s)\n", identifier, list(
optional_formal_argument_list, ", " ) );
    }
    else
    {
        print_level( level );
        margin_printf( "SUBROUTINE %s()\n", identifier );
    }
} /* subroutine_statement */
```

FILE: ctimer/statement/unconditional\_go\_to\_statement.c

```
/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */
```

```
extern int level;
```

```
void unconditional_go_to_statement( label )
register char *label;
{
    print_level( level );
    margin_printf( "GO TO %s\n", label );
} /* unconditional_go_to_statement */
```

FILE: ctimer/statement/write\_statement.c

```
/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */
```

```
extern int level;
extern char *list();
```

```
void write_statement( control_list, optional_io_list )
register char *control_list;
register char *optional_io_list;
{
    if ( optional_io_list != 0 )
    {
        print_level( level );
        margin_printf( "WRITE %s %s\n", list( control_list, ", " ), list(
optional_io_list, ", " ) );
    }
    else
    {
        print_level( level );
        margin_printf( "WRITE (%s)\n", list( control_list, ", " ) );
    }
} /* write_statement */
```

## 11. Appendix F: declare program source

FILE: declare/Makefile

```
#
# Copyright 1991
# Georgia Institute of Technology
# Computer Engineering Research Laboratory
# Author: Stephen R. Wachtel
#

default:    declare

CC = cc -g
INCLUDE = include
CFLAGS = -I$(INCLUDE)
LIBRARY = statement/statement.a library/library.a

OBJECTS = \
$(INCLUDE)/grammar.h \
*grammar.[co] \
*scanner.[co] \
yytrace.[co] \
y.output

PROGRAMS = \
*declare

grammar.c: grammar.y
yacc -dv grammar.y
mv y.tab.h $(INCLUDE)/grammar.h
mv y.tab.c grammar.c

scanner.c: scanner.l
lex -vt scanner.l | sed 's/getc/yygetc/' >scanner.c

scanner.o: scanner.c $(INCLUDE)/grammar.h
$(CC) $(CFLAGS) -c scanner.c

grammar.o: grammar.c
$(CC) $(CFLAGS) -c grammar.c

declare:  grammar.o scanner.o $(LIBRARY)
$(CC) -o declare grammar.o scanner.o $(LIBRARY)

sgrammar.c: grammar.c yytoken.awk
awk -f yytoken.awk <grammar.c >sggrammar.c

sggrammar.o: sggrammar.c
$(CC) $(CFLAGS) -c sggrammar.c

sdeclare: sggrammar.o scanner.o $(LIBRARY)
$(CC) -o sdeclare sggrammar.o scanner.o $(LIBRARY)

dscanner.c: scanner.c
cp scanner.c dscanner.c

dscanner.o: dscanner.c $(INCLUDE)/grammar.h
$(CC) $(CFLAGS) -DDEBUG -c dscanner.c

ddeclare: grammar.o dscanner.o $(LIBRARY)
$(CC) -o ddeclare grammar.o dscanner.o $(LIBRARY)

tgrammar.c: grammar.c
sed 's/yystack:/& yytrace(yystate);/' <grammar.c >tgrammar.c

tgrammar.o: tgrammar.c
$(CC) $(CFLAGS) -c tgrammar.c
```



```
tdeclare: tgrammar.o scanner.o yytrace.o $(LIBRARY)
$(CC) -o tdeclare tgrammar.o scanner.o yytrace.o $(LIBRARY)
```

```
yytrace.c: grammar.c yytrace.awk
awk -f yytrace.awk <y.output >yytrace.c
```

```
yytrace.o: yytrace.c
$(CC) $(CFLAGS) -c yytrace.c
```

```
clean:
cd statement; make clean
cd library; make clean
rm -f $(PROGRAMS) $(OBJECTS)
```

```
FILE: declare/grammar.y
```

```
/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */
```

```
/*
 * FORTRAN 77
 */
```

```
%token RW_AND
%token RW_ASSIGN
%token RW_BACKSPACE
%token RW_BLOCK_DATA
%token RW_CALL
%token RW_CHARACTER
%token RW_CLOSE
%token RW_COMMON
%token RW_COMPLEX
%token RW_CONTINUE
%token RW_DATA
%token RW_DIMENSION
%token RW_DO
%token RW_DOUBLE_PRECISION
%token RW_ELSE
%token RW_ELSE_IF
%token RW_END
%token RW_END IF
%token RW_ENDFILE
%token RW_ENTRY
%token RW_EQ
%token RW_EQUIVALENCE
%token RW_EQV
%token RW_EXTERNAL
%token RW_FALSE
%token RW_FORMAT
%token RW_FUNCTION
%token RW_GE
%token RW_GO_TO
%token RW_GT
%token RW_IF
%token RW_IMPLICIT
%token RW_INCLUDE
%token RW_INQUIRE
%token RW_INTEGER
%token RW_INTRINSIC
%token RW_LE
%token RW_LOGICAL
%token RW_LT
%token RW_NAMelist
%token RW_NE
%token RW_NEQV
%token RW_NOT
%token RW_OPEN
%token RW_OR
%token RW_PARAMETER
%token RW_PAUSE
%token RW_PRINT
```

```

%token RW_PROGRAM
%token RW_READ
%token RW_REAL
%token RW_RETURN
%token RW_REWIND
%token RW_SAVE
%token RW_STOP
%token RW_SUBROUTINE
%token RW_THEN
%token RW_TO
%token RW_TRUE
%token RW_WRITE
%token RW_UNDEFINED

%token COMMENT
%token CONCATENATE
%token DOUBLE PRECISION
%token EXPONENTIATE
%token HOLLERITH
%token IDENTIFIER
%token INTEGER
%token LABEL
%token REAL
%token STRING

%left ','
%nonassoc ':'
%right '='
%left RW_EQV RW_NEQV
%left RW_OR
%left RW_AND
%left RW_NOT
%nonassoc RW_EQ RW_NE RW_LT RW_LE RW_GT RW_GE
%left CONCATENATE
%left '+' '-'
%left '*' '/'
%right EXPONENTIATE
%left SIGN

%{
typedef char *POINTER;
#define YYSTYPE POINTER

#include "list.h"
#include "attribute.h"

extern POINTER duplicate( );
extern POINTER list( );
extern POINTER merge( );
extern POINTER type( );
%}

%%

program:
    optional_statement_list
    {
        summary( );
    }
    ;

optional_statement_list:
    /* NULL */
    |
        statement_list
    ;

statement_list:
    statement
    |
        statement_list statement
    ;

```

```

statement:
    comment_statement
    |
    label unlabeled_statement
    ;

comment_statement:
    COMMENT
    ;

label:
    LABEL
    ;

unlabeled_statement:
    include_statement
    |
    program_statement
    |
    block_data_statement
    |
    function_statement
    |
    subroutine_statement
    |
    entry_statement
    |
    end_statement
    |
    specification_statement
    |
    executable_statement
    |
    format_statement
    ;

include_statement:
    RW_INCLUDE character_constant
    ;

program_statement:
    RW_PROGRAM program_identifier
    {
        program_statement( $2 );
    }
    ;

program_identifier:
    IDENTIFIER
    {
        $$ = $1;
    }
    ;

block_data_statement:
    RW_BLOCK_DATA block_data_identifier
    {
        block_data_statement( $2 );
    }
    ;

block_data_identifier:
    IDENTIFIER
    {
        $$ = $1;
    }
    ;

function_statement:
    RW_FUNCTION function_identifier optional_formal_argument_list
    {
        function_statement( 0, $2, $3 );
    }

```

```

    }
    |
    type RW_FUNCTION function_identifier optional_formal_argument_list
    {
        function_statement( $1, $3, $4 );
    }
    ;

function_identifier:
    IDENTIFIER
    {
        $$ = $1;
    }
    ;

subroutine_statement:
    RW_SUBROUTINE subroutine_identifier
    {
        subroutine_statement( $2, 0 );
    }
    |
    RW_SUBROUTINE subroutine_identifier optional_formal_argument_list
    {
        subroutine_statement( $2, $3 );
    }
    ;

subroutine_identifier:
    IDENTIFIER
    {
        $$ = $1;
    }
    ;

entry_statement:
    RW_ENTRY entry_identifier
    |
    RW_ENTRY entry_identifier optional_formal_argument_list
    ;

entry_identifier:
    IDENTIFIER
    {
        $$ = $1;
    }
    ;

optional_formal_argument_list:
    '(' ')'
    {
        $$ = 0;
    }
    |
    '(' formal_argument_list ')'
    {
        $$ = $2;
    }
    ;

formal_argument_list:
    formal_argument
    {
        $$ = merge( "%s", $1 );
    }
    |
    formal_argument_list ',' formal_argument
    {
        $$ = merge( "%s%s", $1, $3 );
    }
    ;

formal_argument:
    IDENTIFIER

```

```

    {
        $$ = $1;
    }
    |
    formal_argument_alternate_return
    {
        $$ = $1;
    }
    ;

formal_argument_alternate_return:
    '*'
    {
        $$ = duplicate( "*" );
    }
    ;

end_statement:
    RW_END
    {
        end_statement( );
    }
    ;

specification_statement:
    external_statement
    |
    intrinsic_statement
    |
    parameter_statement
    |
    dimension_statement
    |
    declaration_statement
    |
    save_statement
    |
    common_statement
    |
    equivalence_statement
    |
    implicit_statement
    |
    data_statement
    |
    namelist_statement
    ;

external_statement:
    RW_EXTERNAL external_list
    ;

external_list:
    external
    {
        $$ = merge( "%s", $1 );
    }
    |
    external_list ',' external
    {
        $$ = merge( "%s%s", $1, $3 );
    }
    ;

external:
    IDENTIFIER
    {
        $$ = $1;
    }
    ;

intrinsic_statement:
    RW_INTRINSIC intrinsic_list
    ;

```

```

intrinsic_list:
  intrinsic
  {
    $$ = merge( "%s", $1 );
  }
  |
  intrinsic_list ',' intrinsic
  {
    $$ = merge( "%s{%s}", $1, $3 );
  }
  ;

intrinsic:
  IDENTIFIER
  {
    $$ = $1;
  }
  ;

parameter_statement:
  RW_PARAMETER '(' parameter_list ')'
  {
    parameter_statement( $3 );
  }
  ;

parameter_list:
  parameter
  {
    $$ = merge( "%s", $1 );
  }
  |
  parameter_list ',' parameter
  {
    $$ = merge( "%s{%s}", $1, $3 );
  }
  ;

parameter:
  IDENTIFIER '=' expression
  {
    $$ = merge( "%s{%s}", $1, $3 );
  }
  ;

dimension_statement:
  RW_DIMENSION dimension_list
  {
    dimension_statement( $2 );
  }
  ;

dimension_list:
  dimension
  {
    $$ = merge( "%s", $1 );
  }
  |
  dimension_list ',' dimension
  {
    $$ = merge( "%s{%s}", $1, $3 );
  }
  ;

dimension:
  IDENTIFIER '(' subscript_list ')'
  {
    $$ = merge( "%s{%s}", $1, $3 );
  }
  ;

```

```

subscript_list:
    subscript
    {
        $$ = merge( "{%s}", $1 );
    }
    |
    subscript_list ',' subscript
    {
        $$ = merge( "%s{%s}", $1, $3 );
    }
    ;

```

```

subscript:
    upper_bound
    {
        $$ = $1;
    }
    |
    lower_bound ':' upper_bound
    {
        $$ = merge( "%s:%s", $1, $3 );
    }
    ;

```

```

lower_bound:
    expression
    {
        $$ = $1;
    }
    ;

```

```

upper_bound:
    lower_bound
    {
        $$ = $1;
    }
    |
    upper_bound_adjustable
    {
        $$ = $1;
    }
    ;

```

```

upper_bound_adjustable:
    '*'
    {
        $$ = duplicate( "*" );
    }
    ;

```

```

declaration_statement:
    type declaration_list
    {
        declaration_statement( $1, $2 );
    }
    ;

```

```

declaration_list:
    declaration
    {
        $$ = merge( "{%s}", $1 );
    }
    |
    declaration_list ',' declaration
    {
        $$ = merge( "%s{%s}", $1, $3 );
    }
    ;

```

```

declaration:
    IDENTIFIER
    {
        $$ = merge( "{%s}", $1 );
    }
    ;

```

```

| IDENTIFIER '(' subscript_list ')'
| {
|     $$ = merge( "(%s){%s}", $1, $3 );
| }
;

type:
type_name optional_type_length
| {
|     $$ = type( $1, $2 );
| }
;

type_name:
RW_CHARACTER
| {
|     $$ = duplicate( "CHARACTER" );
| }
|
RW_COMPLEX
| {
|     $$ = duplicate( "COMPLEX" );
| }
|
RW_DOUBLE_PRECISION
| {
|     $$ = duplicate( "DOUBLE_PRECISION" );
| }
|
RW_INTEGER
| {
|     $$ = duplicate( "INTEGER" );
| }
|
RW_LOGICAL
| {
|     $$ = duplicate( "LOGICAL" );
| }
|
RW_REAL
| {
|     $$ = duplicate( "REAL" );
| }
|
RW_UNDEFINED
| {
|     $$ = duplicate( "UNDEFINED" );
| }
;

optional_type_length:
/* NULL */
| {
|     $$ = 0;
| }
|
type_length
| {
|     $$ = $1;
| }
;

type_length:
/* INTEGER
| {
|     $$ = $2;
| }
|
/* type_length_adjustable
| {
|     $$ = $2;
| }
;

type_length_adjustable:

```



```

    (' '*' ')
    {
        $$ = duplicate( "(" ) ;
    }
;

save_statement:
    RW_SAVE optional_save_list
;

optional_save_list:
    /* NULL */
    {
        $$ = 0;
    }
    |
    save_list
    {
        $$ = $1;
    }
;

save_list:
    save
    {
        $$ = merge( "({s})", $1 );
    }
    |
    save_list ',' save
    {
        $$ = merge( "%s({s})", $1, $3 );
    }
;

save:
    IDENTIFIER
    {
        $$ = $1;
    }
    |
    common_name
    {
        $$ = $1;
    }
;

common_statement:
    RW_COMMON optional_common_name common_variable_list
    {
        common_statement( $2, $3 );
    }
;

optional_common_name:
    /* NULL */
    {
        $$ = 0;
    }
    |
    common_name
    {
        $$ = $1;
    }
;

common_name:
    '/' optional_identifier '/'
    {
        $$ = $2;
    }
;

optional_identifier:

```

```

        /* NULL */
        {
            $$ = 0;
        }
    IDENTIFIER
    {
        $$ = $1;
    }
;

common_variable_list:
    common_variable
    {
        $$ = merge( "${s}", $1 );
    }
    |
    common_variable_list ',' common_variable
    {
        $$ = merge( "${s}${s}", $1, $3 );
    }
;

common_variable:
    IDENTIFIER
    {
        $$ = merge( "${s}", $1 );
    }
    |
    IDENTIFIER '(' subscript_list ')'
    {
        $$ = merge( "${s}${s}", $1, $3 );
    }
;

equivalence_statement:
    RW_EQUIVALENCE equivalence_list
    {
        equivalence_statement(. $2 );
    }
;

equivalence_list:
    equivalence
    {
        $$ = merge( "${s}", $1 );
    }
    |
    equivalence_list ',' equivalence
    {
        $$ = merge( "${s}${s}", $1, $3 );
    }
;

equivalence:
    '(' equivalence_variable_list ')'
    {
        $$ = $2;
    }
;

equivalence_variable_list:
    equivalence_variable
    {
        $$ = merge( "${s}", $1 );
    }
    |
    equivalence_variable_list ',' equivalence_variable
    {
        $$ = merge( "${s}${s}", $1, $3 );
    }
;

equivalence_variable:

```

```

IDENTIFIER
{
    $$ = merge( "{%s}", $1 );
}
|
IDENTIFIER '(' subscript_list ')'
{
    $$ = merge( "{%s}{%s}", $1, $3 );
}
;

implicit_statement:
    RW_IMPLICIT type '(' implicit_list ')'
    {
        implicit_statement( $2, $4 );
    }
;

implicit_list:
    implicit
    {
        $$ = merge( "{%s}", $1 );
    }
    |
    implicit_list ',' implicit
    {
        $$ = merge( "%s{%s}", $1, $3 );
    }
;

implicit:
    IDENTIFIER
    {
        $$ = merge( "{%s}", $1 );
    }
    |
    IDENTIFIER '--' IDENTIFIER
    {
        $$ = merge( "{%s}{%s}", $1, $3 );
    }
;

namelist_statement:
    RW_NAMELIST namelist_name namelist_list
;

namelist_name:
    '/' IDENTIFIER '/'
    {
        $$ = $2;
    }
;

namelist_list:
    namelist
    {
        $$ = merge( "{%s}", $1 );
    }
    |
    namelist_list ',' namelist
    {
        $$ = merge( "%s{%s}", $1, $3 );
    }
;

namelist:
    IDENTIFIER
    {
        $$ = $1;
    }
;

data_statement:

```

```

        RW_DATA data_list
        {
            data_statement( $2 );
        }
    ;

data_list:
    data
    {
        $$ = merge( "{%s}", $1 );
    }
    |
    data_list optional_comma data
    {
        $$ = merge( "%s{%s}", $1, $3 );
    }
    ;

data:
    data_variable '/' data_constant_list '/'
    {
        $$ = merge( "%s{%s}", $1, $3 );
    }
    ;

data_variable:
    variable
    {
        $$ = $1;
    }
    |
    data_implied_do_list
    {
        $$ = $1;
    }
    ;

data_implied_do_list:
    '(' data_variable ',' IDENTIFIER '=' expression_list ')'
    {
        add_list( 0, $4, 0, 0, IMPLICIT | LOCAL | VARIABLE )->number++;
        $$ = $2;
    }
    ;

data_constant_list:
    data_constant
    {
        $$ = merge( "{%s}", $1 );
    }
    |
    data_constant_list ',' data_constant
    {
        $$ = merge( "%s{%s}", $1, $3 );
    }
    ;

data_constant:
    data_initialization
    {
        $$ = $1;
    }
    |
    IDENTIFIER '**' data_initialization
    {
        $$ = merge( "%s * %s", $1, $3 );
    }
    |
    INTEGER '**' data_initialization
    {
        $$ = merge( "%s * %s", $1, $3 );
    }
    ;

```

data\_initialization:

```

IDENTIFIER
{
    $$ = $1;
}
|
character_constant
{
    $$ = $1;
}
|
logical_constant
{
    $$ = $1;
}
|
signed_numerical_constant
{
    $$ = $1;
}
;

```

signed\_numerical\_constant:

```

numerical_constant
{
    $$ = $1;
}
|
'+' numerical_constant %prec SIGN
{
    $$ = merge( "+%s", $2 );
}
|
'-' numerical_constant %prec SIGN
{
    $$ = merge( "-%s", $2 );
}
;

```

expression:

```

parenthesis_expression
{
    $$ = $1;
}
|
simple_expression
{
    $$ = $1;
}
;

```

parenthesis\_expression:

```

'(' expression ')'
{
    $$ = merge( "( %s )", $2 );
}
;

```

simple\_expression:

```

variable
{
    $$ = $1;
}
|
constant
{
    $$ = $1;
}
|
arithmetic_expression
{
    $$ = $1;
}
|
character_expression
{
    $$ = $1;
}
;

```

```

    }
    relational_expression
    {
        $$ = $1;
    }
    logical_expression
    {
        $$ = $1;
    }
    unary_expression
    {
        $$ = $1;
    }
;

variable:
    IDENTIFIER
    {
        add_list( 0, $1, 0, 0, IMPLICIT | LOCAL | VARIABLE )->number++;
        $$ = $1;
    }
    IDENTIFIER string_subset
    {
        add_list( 0, $1, 0, 0, IMPLICIT | LOCAL | VARIABLE )->number++;
        $$ = merge( "%s%s", $1, $2 );
    }
    array
    {
        $$ = $1;
    }
;

array:
    IDENTIFIER '(' optional_expression_list ')'
    {
        if ( !array( $1 ) )
        {
            add_list( 0, $1, 0, 0, IMPLICIT | GLOBAL | VARIABLE | FUNCTION )->number++;
        }
        else
        {
            add_list( 0, $1, 0, 0, IMPLICIT | LOCAL | VARIABLE | ARRAY )->number++;
        }
        $$ = merge( "%s( %s )", $1, list( $3, " ", " ) );
    }
    IDENTIFIER '(' optional_expression_list ')' string_subset
    {
        if ( !array( $1 ) )
        {
            add_list( 0, $1, 0, 0, IMPLICIT | GLOBAL | VARIABLE | FUNCTION )->number++;
        }
        else
        {
            add_list( 0, $1, 0, 0, IMPLICIT | LOCAL | VARIABLE | ARRAY )->number++;
        }
        $$ = merge( "%s( %s )%s", $1, list( $3, " ", " ), $5 );
    }
;

optional_expression_list:
    /* NULL */
    {
        $$ = 0;
    }
    expression_list
    {
        $$ = $1;
    }
;

```

```

expression_list:
    expression
    {
        $$ = merge( "{%s}", $1 );
    }
    |
    expression_list ',' expression
    {
        $$ = merge( "%s{%s}", $1, $3 );
    }
    ;

string_subset:
    '(' optional_expression ':' optional_expression ')'
    {
        $$ = merge( "( %s : %s )", $2, $4 );
    }
    ;

optional_expression:
    /* NULL */
    {
        $$ = 0;
    }
    |
    expression
    {
        $$ = $1;
    }
    ;

constant:
    character_constant
    {
        $$ = $1;
    }
    |
    logical_constant
    {
        $$ = $1;
    }
    |
    numerical_constant
    {
        $$ = $1;
    }
    ;

character_constant:
    HOLLERITH
    {
        $$ = $1;
    }
    |
    STRING
    {
        $$ = $1;
    }
    ;

logical_constant:
    RW_FALSE
    {
        $$ = duplicate( ".FALSE." );
    }
    |
    RW_TRUE
    {
        $$ = duplicate( ".TRUE." );
    }
    ;

numerical_constant:
    DOUBLE_PRECISION
    {

```

```

        $$ = $1;
    }
    |
    INTEGER
    {
        $$ = $1;
    }
    |
    REAL
    {
        $$ = $1;
    }
    ;

arithmetic_expression:
    expression '+' expression %prec '+'
    {
        $$ = merge( "%s + %s", $1, $3 );
    }
    |
    expression '-' expression %prec '-'
    {
        $$ = merge( "%s - %s", $1, $3 );
    }
    |
    expression '*' expression %prec '*'
    {
        $$ = merge( "%s * %s", $1, $3 );
    }
    |
    expression '/' expression %prec '/'
    {
        $$ = merge( "%s / %s", $1, $3 );
    }
    |
    expression EXPONENTIATE expression %prec EXPONENTIATE
    {
        $$ = merge( "%s ** %s", $1, $3 );
    }
    ;

character_expression:
    expression '/' '/' expression %prec CONCATENATE
    {
        $$ = merge( "%s // %s", $1, $4 );
    }
    ;

relational_expression:
    expression RW_EQ expression %prec RW_EQ
    {
        $$ = merge( "%s .EQ. %s", $1, $3 );
    }
    |
    expression RW_NE expression %prec RW_NE
    {
        $$ = merge( "%s .NE. %s", $1, $3 );
    }
    |
    expression RW_LT expression %prec RW_LT
    {
        $$ = merge( "%s .LT. %s", $1, $3 );
    }
    |
    expression RW_LE expression %prec RW_LE
    {
        $$ = merge( "%s .LE. %s", $1, $3 );
    }
    |
    expression RW_GT expression %prec RW_GT
    {
        $$ = merge( "%s .GT. %s", $1, $3 );
    }
    |
    expression RW_GE expression %prec RW_GE
    {
        $$ = merge( "%s .GE. %s", $1, $3 );
    }
    ;

```



```

;

logical_expression:
    expression RW_AND expression %prec RW_AND
    {
        $$ = merge( "%s .AND. %s", $1, $3 );
    }
    |
    expression RW_OR expression %prec RW_OR
    {
        $$ = merge( "%s .OR. %s", $1, $3 );
    }
    |
    expression RW_EQV expression %prec RW_EQV
    {
        $$ = merge( "%s .EQV. %s", $1, $3 );
    }
    |
    expression RW_NEQV expression %prec RW_NEQV
    {
        $$ = merge( "%s .NEQV. %s", $1, $3 );
    }
;

unary_expression:
    '+' expression %prec SIGN
    {
        $$ = merge( "+%s", $2 );
    }
    |
    '-' expression %prec SIGN
    {
        $$ = merge( "-%s", $2 );
    }
    |
    RW_NOT expression %prec RW_NOT
    {
        $$ = merge( ".NOT. %s", $2 );
    }
;

executable_statement:
    do_statement
    |
    logical_if_statement
    |
    block_if_statement
    |
    else_statement
    |
    else_if_statement
    |
    end_if_statement
    |
    subset_executable_statement
;

do_statement:
    RW_DO optional_integer IDENTIFIER '=' expression_list
    {
        do_statement( $2, $3, $5 );
    }
;

optional_integer:
    /* NULL */
    {
        $$ = 0;
    }
    |
    INTEGER
    {
        $$ = $1;
    }
;

```

```
logical_if_statement:
    if_expression subset_executable_statement
    ;
```

```
if_expression:
    RW_IF '(' expression ')'
    ;
```

```
block_if_statement:
    RW_IF '(' expression ')' RW_THEN
    ;
```

```
else_statement:
    RW_ELSE
    ;
```

```
else_if_statement:
    RW_ELSE_IF '(' expression ')' RW_THEN
    ;
```

```
end_if_statement:
    RW_END_IF
    ;
```

```
subset_executable_statement:
    assignment_statement
    |
    assign_statement
    |
    arithmetic_if_statement
    |
    continue_statement
    |
    call_statement
    |
    return_statement
    |
    unconditional_go_to_statement
    |
    computed_go_to_statement
    |
    assigned_go_to_statement
    |
    stop_statement
    |
    pause_statement
    |
    io_statement
    ;
```

```
assignment_statement:
    variable '=' expression
    ;
```

```
assign_statement:
    RW_ASSIGN INTEGER RW_TO IDENTIFIER
    ;
```

```
arithmetic_if_statement:
    RW_IF '(' expression ')' integer_list
    ;
```

```
continue_statement:
    RW_CONTINUE
    ;
```

```
call_statement:
    RW_CALL IDENTIFIER
    ;
```

```

        RW_CALL IDENTIFIER optional_actual_argument_list
    ;

optional_actual_argument_list:
    '(' ')'
    {
        $$ = 0;
    }
    |
    '(' actual_argument_list ')'
    {
        $$ = $2;
    }
    ;

actual_argument_list:
    actual_argument
    {
        $$ = merge( "%s", $1 );
    }
    |
    actual_argument_list ',' actual_argument
    {
        $$ = merge( "%s%s", $1, $3 );
    }
    ;

actual_argument:
    expression
    {
        $$ = $1;
    }
    |
    actual_argument_alternate_return
    {
        $$ = $1;
    }
    ;

actual_argument_alternate_return:
    '*' INTEGER
    {
        $$ = merge( "**%s", $2 );
    }
    ;

return_statement:
    RW_RETURN optional_expression
    ;

unconditional_go_to_statement:
    RW_GO_TO INTEGER
    ;

computed_go_to_statement:
    RW_GO_TO '(' integer_list ')' optional_comma expression
    ;

assigned_go_to_statement:
    RW_GO_TO IDENTIFIER
    |
    RW_GO_TO IDENTIFIER optional_comma '(' integer_list ')'
    ;

optional_comma:
    /* NULL */
    |
    ','
    ;

integer_list:

```

```

        INTEGER
        {
            $$ = merge( "{%s}", $1 );
        }
    |
    integer_list ',' INTEGER
    {
        $$ = merge( "%s{%s}", $1, $3 );
    }
;

pause_statement:
    RW_PAUSE optional_expression
;

stop_statement:
    RW_STOP optional_expression
;

io_statement:
    open_statement
    |
    close_statement
    |
    inquire_statement
    |
    read_statement
    |
    write_statement
    |
    print_statement
    |
    backspace_statement
    |
    rewind_statement
    |
    endfile_statement
;

open_statement:
    RW_OPEN '(' control_information_list ')'
;

close_statement:
    RW_CLOSE '(' control_information_list ')'
;

inquire_statement:
    RW_INQUIRE '(' control_information_list ')'
;

read_statement:
    RW_READ '(' control_information_list ')' optional_io_list
    |
    RW_READ control
    |
    RW_READ control ',' io_list
;

write_statement:
    RW_WRITE '(' control_information_list ')' optional_io_list
;

print_statement:
    RW_PRINT control
    |
    RW_PRINT control ',' io_list
;

backspace_statement:
    RW_BACKSPACE '(' control_information_list ')'

```

```

:
:   RW_BACKSPACE control
:

rewind_statement:
-   RW_REWIND '(' control_information_list ')'
:
:   RW_REWIND control
:

endfile_statement:
-   RW_ENDFILE '(' control_information_list ')'
:
:   RW_ENDFILE control
:

control_information_list:
-   control_information
-   {
-       $$ = merge( "{%s}", $1 );
-   }
:
-   control_information_list ',' control_information
-   {
-       $$ = merge( "%s{%s}", $1, $3 );
-   }
:

control_information:
-   control
-   {
-       $$ = $1;
-   }
:
-   IDENTIFIER '=' expression
-   {
-       $$ = merge( "%s = %s", $1, $3 );
-   }
:

control:
-   variable
-   {
-       $$ = $1;
-   }
:
-   constant
-   {
-       $$ = $1;
-   }
:
-   '*'
-   {
-       $$ = duplicate( "*" );
-   }
:

optional_io_list:
-   /* NULL */
-   {
-       $$ = 0;
-   }
:
-   io_list
-   {
-       $$ = $1;
-   }
:

io_list:
-   io
-   {
-       $$ = merge( "{%s}", $1 );
-   }

```

```

    io_list ',' io
    {
        $$ = merge( "%s{%s}", $1, $3 );
    }
;

io:
    expression
    {
        $$ = $1;
    }
    |
    io_implied_do_list
    {
        $$ = $_;
    }
;

io_implied_do_list:
    '(' io_list ',' IDENTIFIER '=' expression_list ')'
    {
        add_list( 0, $4, 0, 0, IMPLICIT | LOCAL | VARIABLE )->number++;
        $$ = merge( "( %s, %s = %s )", list( $2, " ", " ), $4, list( $6, " ", " ) );
    }
;

format_statement:
    RW_FORMAT
;

%%

FILE: declare/include/attribute.h

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#define IMPLICIT 0x0000
#define EXPLICIT 0x0001

#define LOCAL 0x0000
#define GLOBAL 0x0002

#define VARIABLE 0x0000
#define CONSTANT 0x0004

#define ARRAY 0x0010
#define COMMON 0x0020

#define FORMAL_ARGUMENT 0x0100
#define EQUIVALENCE 0x0200

#define PROGRAM 0x1000
#define FUNCTION 0x2000

extern char *attribute_name( );

FILE: declare/include/list.h

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```
#define LIST struct list_type
LIST
{
    char *identifier;
    int attribute;
    char *type;
    char *subscript_list;
    char *data;
    int number;
    LIST *next;
};
```

```
extern LIST *end_list();
extern LIST *add_end_list();
extern LIST *find_list();
extern LIST *add_list();
extern void update_list();
extern LIST *delete_list();
```

FILE: declare/include/option.h

```
/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */
```

```
#define INITIALIZE 1
extern int option;
```

FILE: declare/include/table.h

```
/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */
```

```
#define TABLE struct table_type
TABLE
{
    char *identifier;
    int attribute;
    LIST *list;
};
```

```
extern void initialize_table();
extern int add_table();
extern int find_table();
```

```
#define NUMBER_TABLE 250
extern TABLE table[ NUMBER_TABLE ];
extern int number_table;
```

FILE: declare/library/Makefile

```
#
# Copyright 1991
# Georgia Institute of Technology
# Computer Engineering Research Laboratory
# Author: Stephen R. Wachtel
#
```

```
CC = cc -g
INCLUDE = ../include
CFLAGS = -I$(INCLUDE)
```

```
LIBRARY = library.a
```

```
OBJECTS = \
  array.o \
  attribute_name.o \
  count.o \
  duplicate.o \
  hollerith.o \
  implicit.o \
  implicit_data.o \
  link_list.o \
  list.o \
  main.o \
  merge.o \
  non_blank.o \
  parse.o \
  split.o \
  summary.o \
  table.o \
  type.o \
  uppercase.o \
  yyerror.o \
  yygetc.o \
  yywrap.o
```

```
$(LIBRARY):$(OBJECTS)
  ar crv $(LIBRARY) $(OBJECTS)
  ranlib $(LIBRARY)
```

```
.SUFFIXES: .c .o
.c.o:
  $(CC) -c $(CFLAGS) $<
```

```
clean:
  rm -f $(LIBRARY) $(OBJECTS)
```

```
FILE: declare/library/array.c
```

```
/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */
```

```
#include <stdio.h>
#include "list.h"
#include "table.h"
#include "attribute.h"
```

```
int array( identifier )
register char *identifier;
{
  register LIST *temporary = find_list( table[ number_table ].list, identifier, ARRAY );

  return( temporary != (LIST *)NULL );
} /* array */
```

```
FILE: declare/library/attribute_name.c
```

```
/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */
```

```
#include <string.h>
#include "attribute.h"
```



```

char *attribute_name( attribute )
register int attribute;
{
    static char name[ 256 ];
    strcpy( name, "" );

    if ( ( attribute & ( IMPLICIT | EXPLICIT ) ) == EXPLICIT )
        strcat( name, " EXPLICIT" );
    else
        strcat( name, " IMPLICIT" );

    if ( ( attribute & ( LOCAL | GLOBAL ) ) == GLOBAL )
        strcat( name, " GLOBAL" );
    else
        strcat( name, " LOCAL" );

    if ( ( attribute & ( CONSTANT | VARIABLE ) ) == CONSTANT )
        strcat( name, " CONSTANT" );
    else
        strcat( name, " VARIABLE" );

    if ( ( attribute & ARRAY ) == ARRAY )
        strcat( name, " ARRAY" );
    if ( ( attribute & COMMON ) == COMMON )
        strcat( name, " COMMON" );

    if ( ( attribute & FORMAL_ARGUMENT ) == FORMAL_ARGUMENT )
        strcat( name, " FORMAL_ARGUMENT" );
    if ( ( attribute & EQUIVALENCE ) == EQUIVALENCE )
        strcat( name, " EQUIVALENCE" );

    if ( ( attribute & PROGRAM ) == PROGRAM )
        strcat( name, " PROGRAM" );
    if ( ( attribute & FUNCTION ) == FUNCTION )
        strcat( name, " FUNCTION" );

    return( name );
} /* attribute_name */

```

FILE: declare/library/count.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```

int count( string, length, c )
register char *string;
register int length;
register char c;
{
    register int c_count = 0;

    while ( length != 0 )
    {
        if ( *string == c )
            c_count++;

        string++;
        length--;
    }

    return( c_count );
} /* count */

```

FILE: declare/library/duplicate.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```

#include <stdio.h>
#include <string.h>
#include <malloc.h>

char *duplicate( string )
register char *string;
{
    register char *temporary = (char *)NULL;

    if ( string != (char *)NULL )
    {
        if ( ( temporary = (char *)malloc( strlen( string ) + 1 ) ) != (char *)NULL )
            strcpy( temporary, string );
        else
            fprintf( stderr, "ERROR: duplicate( %s )\n", string );
    }

    return( temporary );
} /* duplicate */

```

FILE: declare/library/hollerith.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```

#include <stdio.h>

```

```

char *hollerith( string, delimiter )
register char *string;
register char delimiter;
{
    int hollerith_length;
    register int string_length = 0;

    sscanf( string, "%dh", &hollerith_length );

    string[ string_length++ ] = delimiter;
    while ( hollerith_length != 0 )
    {
        if ( ( string[ string_length ] = yyinput( ) ) == '\n' )
        {
            yyunput( string[ string_length ] );
            break;
        }

        string_length++;
        hollerith_length--;
    }
    string[ string_length++ ] = delimiter;

    string[ string_length ] = '\0';
    return( string );
} /* hollerith */

```

FILE: declare/library/implicit.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```

extern char *default_integer;
extern char *default_logical;
extern char *duplicate( );

```

```

static char *implicit_table[ ] =
{
    /* A */ 0,
    /* B */ 0,
    /* C */ 0,
    /* D */ 0,
    /* E */ 0,
    /* F */ 0,
    /* G */ 0,
    /* H */ 0,
    /* I */ 0,
    /* J */ 0,
    /* K */ 0,
    /* L */ 0,
    /* M */ 0,
    /* N */ 0,
    /* O */ 0,
    /* P */ 0,
    /* Q */ 0,
    /* R */ 0,
    /* S */ 0,
    /* T */ 0,
    /* U */ 0,
    /* V */ 0,
    /* W */ 0,
    /* X */ 0,
    /* Y */ 0,
    /* Z */ 0,

    /* ? */ "UNDEFINED"
};

#define IMPLICIT_TABLE ( sizeof( implicit_table ) / sizeof( char * ) )

int offset( c )
register char *c;
{
    #define LOWER_CASE( c ) ( ( c >= 'a' ) && ( c <= 'z' ) )
    if ( LOWER_CASE( c[ 0 ] ) )
        return( c[ 0 ] - 'a' );

    #define UPPER_CASE( c ) ( ( c >= 'A' ) && ( c <= 'Z' ) )
    if ( UPPER_CASE( c[ 0 ] ) )
        return( c[ 0 ] - 'A' );

    return( IMPLICIT_TABLE - 1 );
} /* offset */

char *implicit_type( string )
register char *string;
{
    return( duplicate( implicit_table[ offset( string ) ] ) );
} /* implicit_type */

void type_implicit( string, lower_bound, upper_bound )
register char *string;
register char *lower_bound;
register char *upper_bound;
{
    register int index;

    if ( upper_bound == 0 )
        upper_bound = lower_bound;

    for ( index = offset( lower_bound ); index <= offset( upper_bound ); index++ )
        implicit_table[ index ] = string;
} /* type_implicit */

void implicit_initialize( )
{
    type_implicit( "REAL*4", "A", "H" );
    type_implicit( default_integer, "I", "N" );
    type_implicit( "REAL*4", "O", "Z" );
} /* implicit_initialize */

```

FILE: declare/library/implicit\_data.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include <stdio.h>
#include "list.h"
#include "attribute.h"

extern char *duplicate( );
extern char *parse( );

int length_subscript_list( subscript_list )
register char *subscript_list;
{
    register char *subscript;
    int upper;
    int lower;
    register int length = 1;

    while ( subscript = parse( subscript_list ) )
    {
        if ( sscanf( subscript, "%d:%d", &upper, &lower ) != 2 )
        {
            upper = atoi( subscript );
            lower = 1;
        }

        length *= ( upper - lower + 1 );
    }

    return( length );
} /* length_subscript_list */

char *data_value( type )
register char *type;
{
    int length;

    if ( sscanf( type, "CHARACTER*%d", &length ) == 1 )
        return( " ' ' " );

    if ( sscanf( type, "COMPLEX*%d", &length ) == 1 )
    {
        switch ( length )
        {
            case 8:
                return( "(OE0, OE0)" );

            case 16:
                return( "(OD0, OD0)" );

            }
    }

    if ( sscanf( type, "INTEGER*%d", &length ) == 1 )
    {
        switch ( length )
        {
            case 1:
                return( "0" );

            case 2:
                return( "0" );

            case 4:
                return( "0" );

            }
    }

    if ( sscanf( type, "LOGICAL*%d", &length ) == 1 )
    {

```

```

switch ( length )
{
    case 1:
        return( ".FALSE." );

    case 2:
        return( ".FALSE." );

    case 4:
        return( ".FALSE." );
}

if ( sscanf( type, "REAL*%d", &length ) == 1 )
{
    switch ( length )
    {
        case 4:
            return( "0E0" );

        case 8:
            return( "0D0" );
    }
}

fprintf( stderr, "ERROR: data_value( %s )\n", type );
exit( -1 );
} /* data_value */

char *implicit_data( entry )
register LIST *entry;
{
    static char data[ 256 ];

    if ( ( entry->attribute & ARRAY ) == ARRAY )
        sprintf( data, "%d * %s", length_subscript_list( duplicate( entry->subscript_list
) ), data_value( entry->type ) );
    else
        sprintf( data, "%s", data_value( entry->type ) );

    return( data );
} /* implicit_data */

```

FILE: declare/library/link\_list.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include <stdio.h>
#include <string.h>
#include <malloc.h>
#include "list.h"
#include "table.h"
#include "attribute.h"

extern char *implicit_type();

#define ZERO( a, b ) ( ( a != 0 ) ? a : b )

LIST *end_list( list )
register LIST *list;
{
    if ( list != (LIST *)NULL )
    {
        while ( list->next != (LIST *)NULL )
            list = list->next;
    }

    return( list );
} /* end_list */

```

```

LIST *add_end_list( list, identifier )
register LIST **list;
register char *identifier;
{
    register LIST *temporary = (LIST *)malloc( sizeof( LIST ) );

    temporary->identifier = identifier;
    temporary->type = (char *)NULL;
    temporary->subscript_list = (char *)NULL;
    temporary->data = (char *)NULL;
    temporary->attribute = 0;
    temporary->number = 0;
    temporary->next = (LIST *)NULL;

    if ( *list == (LIST *)NULL )
        *list = temporary;
    else
        end_list( *list )->next = temporary;

    return( temporary );
} /* add_end_list */

LIST *find_list( list, identifier, attribute )
register LIST *list;
register char *identifier;
register int attribute;
{
    register LIST *temporary;

    while ( list != (LIST *)NULL )
    {
        if ( ( list->attribute & COMMON ) == COMMON )
        {
            if ( ( temporary = find_list( table[ find_table( list->identifier ) ].list,
identifier, attribute ) ) != (LIST *)NULL )
                return( temporary );
        }
        else
        {
            if ( ( strcmp( list->identifier, identifier ) == 0 )
&& ( ( list->attribute & attribute ) == attribute ) )
                return( list );
        }

        list = list->next;
    }

    return( (LIST *)NULL );
} /* find_list */

LIST *add_list( structure, identifier, type, subscript_list, attribute )
register char *structure;
register char *identifier;
register char *type;
register char *subscript_list;
register int attribute;
{
    register int table_number;
    register LIST *temporary;

    if ( structure != (char *)NULL )
        table_number = find_table( structure );
    else
        table_number = number_table;

    temporary = find_list( table[ table_number ].list, identifier, attribute & GLOBAL );
    if ( temporary == (LIST *)NULL )
        temporary = add_end_list( table[ table_number ].list, identifier );

    if ( ( attribute & ( IMPLICIT | EXPLICIT ) ) == EXPLICIT )
        temporary->type = type;
    else
        temporary->type = ZERO( temporary->type, implicit_type( identifier ) );

    temporary->subscript_list = ZERO( temporary->subscript_list, subscript_list );

    temporary->attribute |= attribute;
}

```

```

    return( temporary );
} /* add_list */

void update_list( structure )
register char *structure;
{
    register int table_number;
    register LIST *list;

    if ( structure != (char *)NULL )
        table_number = find_table( structure );
    else
        table_number = number_table;

    list = table[ table_number ].list;
    while ( list != (LIST *)NULL )
    {
        if ( ( list->attribute & COMMON ) == COMMON )
            update_list( list->identifier );
        else
        {
            if ( ( list->attribute & EXPLICIT ) != EXPLICIT )
                list->type = implicit_type( list->identifier );
        }

        list = list->next;
    }
} /* update_list */

```

```

LIST *delete_list( structure, identifier )
register char *structure;
register char *identifier;
{
    register int table_number;
    register LIST *last = (LIST *)NULL;
    register LIST *curr;

    if ( structure != (char *)NULL )
        table_number = find_table( structure );
    else
        table_number = number_table;

    curr = table[ table_number ].list;
    while ( curr != (LIST *)NULL )
    {
        if ( strcmp( curr->identifier, identifier ) == 0 )
        {
            if ( last == (LIST *)NULL )
                table[ table_number ].list = curr->next;
            else
                last->next = curr->next;

            break;
        }

        last = curr;
        curr = curr->next;
    }

    return( curr );
} /* delete_list */

```

FILE: declare/library/list.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```

extern char *parse( );
extern char *merge( );

```

```

char *list( input_list, delimiter )
register char *input_list;
register char *delimiter;
{
    register char *output_list;
    register char *list;
    register char *temporary;

    output_list = parse( input_list );
    list = parse( input_list );

    while ( list != (char *)0 )
    {
        temporary = merge( "%s%s%s", output_list, delimiter, list );

        free( output_list );
        free( list );

        output_list = temporary;
        list = parse( input_list );
    }

    return( output_list );
} /* list */

FILE: declare/library/main.c

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include <stdio.h>
#include <string.h>
#include "option.h"

extern FILE *yyin;
extern FILE *yyout;
extern char *default_integer;
extern char *default_logical;

int option = 0;

#define PROGRAM argument[ 0 ]
#define INPUT_FILE argument[ 1 ]
#define OUTPUT_FILE argument[ 2 ]

int main( number_argument, argument )
int number_argument;
char *argument[ ];
{
loop:
    if ( strcmp( argument[ number_argument - 1 ], "--size=2" ) == 0 )
    {
        number_argument--;
        default_integer = "INTEGER*2";
        default_logical = "LOGICAL*2";
        goto loop;
    }

    if ( strcmp( argument[ number_argument - 1 ], "--size=4" ) == 0 )
    {
        number_argument--;
        default_integer = "INTEGER*4";
        default_logical = "LOGICAL*4";
        goto loop;
    }

    if ( strcmp( argument[ number_argument - 1 ], "-initialize=y" ) == 0 )
    {
        number_argument--;
        option |= INITIALIZE;
    }
}

```



```

    goto loop;
}

if ( strcmp( argument[ number_argument - 1 ], "-initialize=n" ) == 0 )
{
    number_argument--;
    option |= INITIALIZE;
    goto loop;
}

implicit_initialize();
initialize_table();

if ( number_argument == 1 )
{
    yyin = stdin;
    yyout = stdout;

    yyparse();
    exit( 0 );
}

if ( number_argument == 3 )
{
    if ( ( yyin = fopen( INPUT_FILE, "r" ) ) == (FILE *)NULL )
    {
        fprintf( stderr, "%s: ERROR - unable to open input file '%s'\n", PROGRAM,
INPUT_FILE );
        exit( -1 );
    }

    if ( ( yyout = fopen( OUTPUT_FILE, "w" ) ) == (FILE *)NULL )
    {
        fprintf( stderr, "%s: ERROR - unable to open output file '%s'\n", PROGRAM,
OUTPUT_FILE );
        exit( -1 );
    }

    yyparse();
    exit( 0 );
}

fprintf( stderr, "usage: %s <input file> <output file> [-size=2 or 4] or [-
initialize=y or n]\n", PROGRAM );
exit( 0 );
} /* main */

FILE: declare/library/merge.c

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include <stdio.h>
#include <string.h>
#include <malloc.h>

#define STRLEN( s ) ( strlen( s ) - 2 )

char *merge( string, a, b, c, d )
register char *string;
register char *a;
register char *b;
register char *c;
register char *d;
{
    register char *temporary = (char *)NULL;

    switch ( count( string, strlen( string ), '%' ) )
    {
        case 0:
            if ( ( temporary = (char *)malloc( strlen( string ) + 1 ) ) != (char *)NULL )
                sprintf( temporary, string );
    }

```

```

        else
            fprintf( stderr, "ERROR: merge( %s )\n", string );
            break;

    case 1:
        if ( ( temporary = (char *)malloc( strlen( string ) + STRLEN( a ) + 1 ) ) !=
(char *)NULL )
            sprintf( temporary, string, a );
        else
            fprintf( stderr, "ERROR: merge( %s, %s )\n", string, a );
            break;

    case 2:
        if ( ( temporary = (char *)malloc( strlen( string ) + STRLEN( a ) + STRLEN( b
) + 1 ) ) != (char *)NULL )
            sprintf( temporary, string, a, b );
        else
            fprintf( stderr, "ERROR: merge( %s, %s, %s )\n", string, a, b );
            break;

    case 3:
        if ( ( temporary = (char *)malloc( strlen( string ) + STRLEN( a ) + STRLEN( b
) + STRLEN( c ) + 1 ) ) != (char *)NULL )
            sprintf( temporary, string, a, b, c );
        else
            fprintf( stderr, "ERROR: merge( %s, %s, %s, %s )\n", string, a, b, c );
            break;

    case 4:
        if ( ( temporary = (char *)malloc( strlen( string ) + STRLEN( a ) + STRLEN( b
) + STRLEN( c ) + STRLEN( d ) + 1 ) ) != (char *)NULL )
            sprintf( temporary, string, a, b, c, d );
        else
            fprintf( stderr, "ERROR: merge( %s, %s, %s, %s, %s )\n", string, a, b, c, d
);
            break;

    default:
        fprintf( stderr, "ERROR: merge( %s )\n", string );
        break;
}

return( temporary );
} /* merge */

```

FILE: declare/library/non\_blank.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include <string.h>

char *non_blank( string )
register char *string;
{
    register int offset;
    register int length;

    length = strlen( string ) - 1;
    while ( ( string[ length ] == ' ' ) && ( string[ length ] != '\0' ) )
        string[ length-- ] = '\0';

    offset = 0;
    while ( ( string[ offset ] == ' ' ) && ( string[ offset ] != '\0' ) )
        string[ offset++ ] = '\0';

    strcpy( string, &string[ offset ] );

    if ( strlen( string ) != 0 )
        return( string );
    else
        return( 0 );
} /* non_blank */

```

FILE: declare/library/parse.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include <string.h>

extern char *duplicate( );

char *parse( list )
register char *list;
{
    register int length = 0;
    register int brace = 0;
    register char *temporary = (char *)0;

    for (;;)
    {
        switch ( list[ length ] )
        {
            case '[':
                brace++;
                break;

            case ']':
                brace--;
                break;

            }

        if ( brace == 0 )
            break;

        length++;
    }

    if ( length != 0 )
    {
        list[ length ] = '\0';
        temporary = duplicate( list + 1 );
        strcpy( list, list + 1 + length );
    }
    else
    {
        if ( list[ length ] != '\0' )
        {
            temporary = duplicate( list );
            list[ length ] = '\0';
        }
    }

    return( temporary );
} /* parse */

```

FILE: declare/library/split.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

char *split( string, number, delimiter )
register char *string;
register int number;
register char delimiter;
{
    register int count = 0;

```

```

register char *c;

c = string;
while ( *c != '\0' )
{
    if ( *c == delimiter )
        count++;

    c++;
}

count /= number;

c = string;
while ( *c != '\0' )
{
    if ( *c == delimiter )
    {
        if ( --count == 0 )
            break;
    }

    c++;
}

*c = '\0';
return( ++c );
} /* split */

FILE: declare/library/summary.c

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include <stdio.h>
#include <string.h>
#include "list.h"
#include "table.h"
#include "attribute.h"
#include "option.h"

extern FILE *yyin;
extern FILE *yyout;
extern char *duplicate( );
extern char *implicit_data( );
extern char *list( );
extern char *split( );

void output_subprogram_statement( file, identifier, attribute )
register FILE *file;
register char *identifier;
register int attribute;
{
    if ( ( attribute & PROGRAM ) == PROGRAM )
    {
        if ( ( attribute & CONSTANT ) == CONSTANT )
            fprintf( file, "\tPROGRAM %s\n", identifier );
        else
            fprintf( file, "\tBLOCK DATA %s\n", identifier );
    }

    if ( ( attribute & FUNCTION ) == FUNCTION )
    {
        if ( ( attribute & CONSTANT ) == CONSTANT )
            fprintf( file, "\tSUBROUTINE %s()\n", identifier );
        else
            fprintf( file, "\tFUNCTION %s()\n", identifier );
    }
} /* output_subprogram_statement */

void output_end_statement( file )

```

```

register FILE *file;
{
    fprintf( file, "\tEND\n" );
} /* output_end_statement */

void output_declaration_statement( file, entry )
register FILE *file;
register LIST *entry;
{
    if ( ( entry->attribute & ( LOCAL | GLOBAL ) ) == LOCAL )
    {
        fprintf( file, "\t%s %s", entry->type, entry->identifier );

        if ( entry->subscript_list != (char *)NULL )
            fprintf( file, "({%s)", list( duplicate( entry->subscript_list ), ", " ) );

        fprintf( file, "\n" );

        if ( ( ( entry->attribute ) & ( VARIABLE | CONSTANT ) ) == CONSTANT )
            fprintf( file, "\tPARAMETER (%s = %s)\n", entry->identifier, entry->data );
    }
} /* output_declaration_statement */

void output_common_statement( file, identifier )
register FILE *file;
register char *identifier;
{
    register LIST *list;
    register char delimiter;

    fprintf( file, "\tCOMMON /%s/", identifier );

    delimiter = ' ';
    list = table[ find table( identifier ) ].list;
    while ( list != (LIST *)NULL )
    {
        fprintf( file, "%c%s", delimiter, list->identifier );
        delimiter = ',';

        list = list->next;
    }
    fprintf( file, "\n" );

    list = table[ find table( identifier ) ].list;
    while ( list != (LIST *)NULL )
    {
        output_declaration_statement( file, list );

        list = list->next;
    }
} /* output_common_statement */

void output_data_statement( file, entry )
register FILE *file;
register LIST *entry;
{
    register int lower;
    register int upper;
    register char *temporary;

    if ( ( entry->attribute & EQUIVALENCE ) == EQUIVALENCE )
        fprintf( file, "** EQUIVALENCE\n" );

    if ( entry->data != (char *)NULL )
    {
        if ( count( entry->data, strlen( entry->data ), ',' ) <= 100 )
            fprintf( file, "\tDATA %s /%s/\n", entry->identifier, entry->data );
        else
        {
            temporary = split( entry->data, 2, ',' );

            lower = 1;
            upper = lower + count( entry->data, strlen( entry->data ), ',' );

            fprintf( file, "\tDATA ( %s(i), i=%d, %d ) /%s/\n", entry->identifier, lower,
upper, entry->data );

            lower = upper + 1;

```

```

        upper = lower + count( temporary, strlen( temporary ), ',' );

        fprintf( file, "\tDATA ( %s(i), i=%d, %d ) /%s/\n", entry->identifier, lower,
upper, temporary );

    }
    else
        fprintf( file, "\tDATA %s /%s/\n", entry->identifier, implicit_data( entry ) );
} /* output_data_statement */

void output_block_data_statement( file )
register FILE *file;
{
    register int table_number;
    register LIST *list;

    output_subprogram_statement( file, "BLKDAT", IMPLICIT | GLOBAL | VARIABLE | PROGRAM );

    for ( table_number = 0; table_number != NUMBER_TABLE; table_number++ )
    {
        if ( table[ table_number ].identifier == (char *)NULL )
            break;

        if ( ( table[ table_number ].attribute & COMMON ) == COMMON )
        {
            fprintf( file, "** COMMON /%s/ DECLARATION\n", table[ table_number ].identifier
);

            output_common_statement( file, table[ table_number ].identifier );
        }

        for ( table_number = 0; table_number != NUMBER_TABLE; table_number++ )
        {
            if ( table[ table_number ].identifier == (char *)NULL )
                break;

            if ( ( table[ table_number ].attribute & COMMON ) == COMMON )
            {
                fprintf( file, "** COMMON /%s/ INITIALIZATION\n", table[ table_number
].identifier );

                list = table[ table_number ].list;
                while ( list != (LIST *)NULL )
                {
                    output_data_statement( file, list );

                    list = list->next;
                }
            }
        }

        output_end_statement( file );
    } /* output_block_data_statement */

void summary( )
{
    register int table_number;
    register LIST *list;
    register int count;

    for ( table_number = 0; table_number != NUMBER_TABLE; table_number++ )
    {
        if ( table[ table_number ].identifier == (char *)NULL )
            break;

        if ( ( table[ table_number ].attribute & COMMON ) != COMMON )
        {
            output_subprogram_statement( yyout, table[ table_number ].identifier, table[
table_number ].attribute );

            count = 0;
            list = table[ table_number ].list;
            while ( list != (LIST *)NULL )
            {
                if ( ( ( list->attribute & COMMON ) != COMMON )
&& ( ( list->attribute & FORMAL_ARGUMENT ) == FORMAL_ARGUMENT ) )
                {

```

```

        if ( ++count == 1 )
            fprintf( yyout, "** FORMAL ARGUMENT DECLARATION\n" );

        output_declaration_statement( yyout, list );
    }

    list = list->next;
}

list = table[ table_number ].list;
while ( list != (LIST *)NULL )
{
    if ( ( list->attribute & COMMON ) == COMMON )
    {
        fprintf( yyout, "** COMMON /%s/ DECLARATION\n", list->identifier );
        output_common_statement( yyout, list->identifier );
    }

    list = list->next;
}

count = 0;
list = table[ table_number ].list;
while ( list != (LIST *)NULL )
{
#ifdef DEBUG
    if ( list->number != 0 )
#endif
        if ( ( ( list->attribute & COMMON ) != COMMON )
            && ( ( list->attribute & FORMAL_ARGUMENT ) != FORMAL_ARGUMENT ) )
        {
            if ( ++count == 1 )
                fprintf( yyout, "** VARIABLE DECLARATION\n" );

            output_declaration_statement( yyout, list );
        }

        list = list->next;
}

if ( ( option & INITIALIZE ) == INITIALIZE )
{
    count = 0;
    list = table[ table_number ].list;
    while ( list != (LIST *)NULL )
    {
        if ( ( ( list->attribute & COMMON ) != COMMON )
            && ( ( list->attribute & FUNCTION ) != FUNCTION )
            && ( ( list->attribute & CONSTANT ) != CONSTANT )
            && ( ( list->attribute & FORMAL_ARGUMENT ) != FORMAL_ARGUMENT ) )
        {
#ifdef DEBUG
            if ( list->number != 0 )
            {
                if ( list->data != (char *)NULL )
                {
                    if ( ++count == 1 )
                        fprintf( yyout, "** INITIALIZED DATA\n" );

                    output_data_statement( yyout, list );
                }
            }
#endif

            list = list->next;
        }
    }
}

if ( ( option & INITIALIZE ) == INITIALIZE )
{
    count = 0;
    list = table[ table_number ].list;
    while ( list != (LIST *)NULL )
    {
        if ( ( ( list->attribute & COMMON ) != COMMON )
            && ( ( list->attribute & FUNCTION ) != FUNCTION )
            && ( ( list->attribute & CONSTANT ) != CONSTANT )
            && ( ( list->attribute & FORMAL_ARGUMENT ) != FORMAL_ARGUMENT ) )
        {
#ifdef DEBUG
            if ( list->number != 0 )

```

```

#endif
        if ( list->data == (char *)NULL )
        {
            if ( ++count == 1 )
                fprintf( yyout, "** UNINITIALIZED DATA\n" );

            output_data_statement( yyout, list );
        }
        list = list->next;
    }
    output_end_statement( yyout );
}

if ( ( option & INITIALIZE ) == INITIALIZE )
    output_block_data_statement( yyout );
} /* summary */

```

FILE: declare/library/table.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include <stdio.h>
#include <string.h>
#include "list.h"
#include "table.h"
#include "attribute.h"

int number_table = 0;
TABLE table[ NUMBER_TABLE ];

void initialize_table( )
{
    register int table_number;

    for ( table_number = 0; table_number != NUMBER_TABLE; table_number++ )
    {
        table[ table_number ].identifier = (char *)NULL;
        table[ table_number ].attribute = 0;
        table[ table_number ].list = (LIST *)NULL;
    }
} /* initialize_table */

int add_table( identifier, attribute )
register char *identifier;
register int attribute;
{
    register int table_number;

    for ( table_number = 0; table_number != NUMBER_TABLE; table_number++ )
    {
        if ( table[ table_number ].identifier == (char *)NULL )
        {
            table[ table_number ].identifier = identifier;
            table[ table_number ].attribute = attribute;
            table[ table_number ].list = (LIST *)NULL;

            return( table_number );
        }
    }

    fprintf( stderr, "ERROR: add_table( %s )\n", identifier );
    exit( -1 );
} /* add_table */

```



```

int find_table( identifier )
register char *identifier;
{
    register int table_number;

    for ( table_number = 0; table_number != NUMBER_TABLE; table_number++ )
    {
        if ( table[ table_number ].identifier == (char *)NULL )
            break;

        if ( strcmp( identifier, table[ table_number ].identifier ) == 0 )
            return( table_number );
    }

    return( -1 );
} /* find_table */

```

FILE: declare/library/type.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include <string.h>

extern char *duplicate( );
extern char *merge( );

char *default_integer = "INTEGER*4";
char *default_logical = "LOGICAL*4";

char *type( type_name, type_length )
register char *type_name;
register char *type_length;
{
    if ( type_length != (char *)0 )
        return( merge( "%s*%s", type_name, type_length ) );

    if ( strcmp( type_name, "CHARACTER" ) == 0 )
        return( duplicate( "CHARACTER*1" ) );

    if ( strcmp( type_name, "COMPLEX" ) == 0 )
        return( duplicate( "COMPLEX*8" ) );

    if ( strcmp( type_name, "DOUBLE_PRECISION" ) == 0 )
        return( duplicate( "REAL*8" ) );

    if ( strcmp( type_name, "INTEGER" ) == 0 )
        return( duplicate( default_integer ) );

    if ( strcmp( type_name, "LOGICAL" ) == 0 )
        return( duplicate( default_logical ) );

    if ( strcmp( type_name, "REAL" ) == 0 )
        return( duplicate( "REAL*4" ) );

    return( duplicate( type_name ) );
} /* type */

```

FILE: declare/library/uppercase.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

char *uppercase( string )

```

```

register char *string;
{
    register int index = 0;

    while ( string[ index ] != '\0' )
    {
        string[ index ] = toupper( string[ index ] );
        index++;
    }

    return( string );
} /* uppercase */

```

FILE: declare/library/yyerror.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include <stdio.h>

extern int yylineno;

void yyerror( string )
register char *string;
{
    fprintf( stderr, "line %d, %s\n", yylineno, string );

    exit( -1 );
} /* yyerror */

```

FILE: declare/library/yygetc.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include <stdio.h>
#include <ctype.h>

extern int yylineno;

int tab( length )
register int length;
{
    while ( length-- != 0 )
        yyunput( ' ' );

    return( ' ' );
} /* tab */

int yygetc( file )
register FILE *file;
{
    int c;
    int column[ 6 ];

loop:
    if ( ( c = getc( file ) ) == '\t' )
        c = tab( 6 );

    if ( c != '\n' )
        return( c );
}

```

```

    if ( ( column[ 0 ] = getc( file ) ) != ' ' )
        goto abort_0;
    if ( ( column[ 1 ] = getc( file ) ) != ' ' )
        goto abort_1;
    if ( ( column[ 2 ] = getc( file ) ) != ' ' )
        goto abort_2;
    if ( ( column[ 3 ] = getc( file ) ) != ' ' )
        goto abort_3;
    if ( ( column[ 4 ] = getc( file ) ) != ' ' )
        goto abort_4;
    if ( isspace( column[ 5 ] = getc( file ) ) )
        goto abort_5;

    yylineno++;
    goto loop;

abort_5:
    if ( column[ 5 ] == '\t' )
        tab( 1 );
    else
    {
        yyunput( column[ 5 ] );
        if ( column[ 5 ] == '\n' )
            yylineno++;
    }

abort_4:
    if ( column[ 4 ] == '\t' )
        tab( 2 );
    else
    {
        yyunput( column[ 4 ] );
        if ( column[ 4 ] == '\n' )
            yylineno++;
    }

abort_3:
    if ( column[ 3 ] == '\t' )
        tab( 3 );
    else
    {
        yyunput( column[ 3 ] );
        if ( column[ 3 ] == '\n' )
            yylineno++;
    }

abort_2:
    if ( column[ 2 ] == '\t' )
        tab( 4 );
    else
    {
        yyunput( column[ 2 ] );
        if ( column[ 2 ] == '\n' )
            yylineno++;
    }

abort_1:
    if ( column[ 1 ] == '\t' )
        tab( 5 );
    else
    {
        yyunput( column[ 1 ] );
        if ( column[ 1 ] == '\n' )
            yylineno++;
    }

abort_0:
    if ( column[ 0 ] == '\t' )
        tab( 6 );
    else
    {
        yyunput( column[ 0 ] );
        if ( column[ 0 ] == '\n' )
            yylineno++;
    }

    return( c );
} /* yygetc */

```

FILE: declare/library/yywrap.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```

int yywrap( )
{
    return( 1 );
} /* yywrap */

```

FILE: declare/scanner.l

```

%{
/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */
%}

```

```

%a 10000
%e 10000
%k 10000
%n 10000
%o 10000
%p 10000

```

```

a  [aA]
b  [bB]
c  [cC]
d  [dD]
e  [eE]
f  [fF]
g  [gG]
h  [hH]
i  [iI]
j  [jJ]
k  [kK]
l  [lL]
m  [mM]
n  [nN]
o  [oO]
p  [pP]
q  [qQ]
r  [rR]
s  [sS]
t  [tT]
u  [uU]
v  [vV]
w  [wW]
x  [xX]
y  [yY]
z  [zZ]

```

```

%{
#include "grammar.h"
extern char *yylval;

```

```

#undef YYLMAX
#define YYLMAX (256*20)

```

```

extern char *duplicate( );
extern char *hollerith( );
extern char *non_blank( );
extern char *uppercase( );
%}

```

%%

```

^[\\*cC].*[\\n] |
^[\\ ]*[\\n] {
#ifdef DEBUG
    ECHO;
#endif
    yyval = duplicate( yytext );
    return( COMMENT );
}

```

```

[\\ ] {
#ifdef DEBUG
    ECHO;
#endif
    /* return( '\\ ' ) */;
}

```

```

[\\&] {
#ifdef DEBUG
    ECHO;
#endif
    return( '\\&' );
}

```

```

[\\()] {
#ifdef DEBUG
    ECHO;
#endif
    return( '\\(' );
}

```

```

[\\)] {
#ifdef DEBUG
    ECHO;
#endif
    return( '\\)' );
}

```

```

[\\*] {
#ifdef DEBUG
    ECHO;
#endif
    return( '\\*' );
}

```

```

[\\*][\\*] {
#ifdef DEBUG
    ECHO;
#endif
    return( EXPONENTIATE );
}

```

```

[\\+] {
#ifdef DEBUG
    ECHO;
#endif
    return( '\\+' );
}

```

```

[\\,] {
#ifdef DEBUG
    ECHO;
#endif
    return( '\\,' );
}

```

```

[\\-] {
#ifdef DEBUG
    ECHO;
#endif
}

```

```

        return( '\-' );
    }

[\.] {
#ifdef DEBUG
    ECHO;
#endif
    return( '\.' );
}

[\/] {
#ifdef DEBUG
    ECHO;
#endif
    return( '/' );
}

[\:] {
#ifdef DEBUG
    ECHO;
#endif
    return( ':' );
}

[\=] {
#ifdef DEBUG
    ECHO;
#endif
    return( '=' );
}

[\n] {
#ifdef DEBUG
    ECHO;
#endif
    /* return( '\n' ) */;
}

[\t] {
#ifdef DEBUG
    ECHO;
#endif
    /* return( '\t' ) */;
}

[\.]{a}{n}{d}[\.] {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_AND );
}

[\.]{e}{q}[\.] {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_EQ );
}

[\.]{e}{q}{v}[\.] {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_EQV );
}

[\.]{f}{a}{l}{s}{e}[\.] {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_FALSE );
}

```

```

    }

[\\.] {g} {e} [\\.] {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_GE );
}

[\\.] {g} {t} [\\.] {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_GT );
}

[\\.] {l} {e} [\\.] {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_LE );
}

[\\.] {l} {t} [\\.] {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_LT );
}

[\\.] {n} {e} [\\.] {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_NE );
}

[\\.] {n} {e} {q} {v} [\\.] {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_NEQV );
}

[\\.] {n} {o} {t} [\\.] {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_NOT );
}

[\\.] {o} {r} [\\.] {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_OR );
}

[\\.] {t} {r} {u} {e} [\\.] {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_TRUE );
}

{a} {s} {s} {i} {g} {n} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_ASSIGN );
}

```

```

{b}{a}{c}{k}{s}{p}{a}{c}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_BACKSPACE );
}

{b}{l}{o}{c}{k}{[ \ ]*(d){a}{t}{a} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_BLOCK_DATA );
}

{c}{a}{l}{l} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_CALL );
}

{c}{h}{a}{r}{a}{c}{t}{e}{r} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_CHARACTER );
}

{c}{l}{o}{s}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_CLOSE );
}

{c}{o}{m}{m}{o}{n} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_COMMON );
}

{c}{o}{m}{p}{l}{e}{x} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_COMPLEX );
}

{c}{o}{n}{t}{i}{n}{u}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_CONTINUE );
}

{d}{a}{t}{a} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_DATA );
}

{d}{i}{m}{e}{n}{s}{i}{o}{n} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_DIMENSION );
}

```



```

{d}{o} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_DO );
}

{d}{o}{u}{b}{l}{e}{\ }*(p){r}{e}{c}{i}{s}{i}{o}{n} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_DOUBLE_PRECISION );
}

{e}{l}{s}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_ELSE );
}

{e}{l}{s}{e}{\ }*(i){f} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_ELSE_IF );
}

{e}{n}{d} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_END );
}

{e}{n}{d}{\ }*(i){f} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_END_IF );
}

{e}{n}{d}{f}{i}{l}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_ENDFILE );
}

{e}{n}{t}{r}{y} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_ENTRY );
}

{e}{q}{u}{i}{v}{a}{l}{e}{n}{c}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_EQUIVALENCE );
}

{e}{x}{t}{e}{r}{n}{a}{l} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_EXTERNAL );
}

```

```
{f}{o}{r}{m}{a}{t}.* {
#ifdef DEBUG
    ECHO;
#endif
    yylval = duplicate( yytext );
    return( RW_FORMAT );
}
```

```
{f}{u}{n}{c}{t}{i}{o}{n} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_FUNCTION );
}
```

```
{g}{o}{\ }*(t){o} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_GO_TO );
}
```

```
{i}{f} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_IF );
}
```

```
{i}{r}{p}{l}{i}{c}{i}{t} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_IMPLICIT );
}
```

```
{i}{n}{c}{l}{u}{d}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_INCLUDE );
}
```

```
{i}{n}{q}{u}{i}{r}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_INQUIRE );
}
```

```
{i}{n}{t}{e}{g}{e}{r} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_INTEGER );
}
```

```
{i}{n}{t}{r}{i}{n}{s}{i}{c} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_INTRINSIC );
}
```

```
{l}{o}{g}{i}{c}{a}{l} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_LOGICAL );
}
```

```
{n}{a}{m}{e}{l}{i}{s}{t} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_NAMELIST );
}
```

```
{o}{p}{e}{n} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_OPEN );
}
```

```
{p}{a}{r}{a}{m}{e}{t}{e}{r} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_PARAMETER );
}
```

```
{p}{a}{u}{s}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_PAUSE );
}
```

```
{p}{r}{i}{n}{t} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_PRINT );
}
```

```
{p}{r}{o}{g}{r}{a}{m} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_PROGRAM );
}
```

```
{r}{e}{a}{d} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_READ );
}
```

```
{r}{e}{a}{l} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_REAL );
}
```

```
{r}{e}{t}{u}{r}{n} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_RETURN );
}
```

```
{r}{e}{w}{i}{n}{d} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_REWIND );
}
```

```
{s}{a}{v}{e} {
```

```

#ifdef DEBUG
    ECHO;
#endif
    return( RW_SAVE );
}

{s}{t}{o}{p} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_STOP );
}

{s}{u}{b}{r}{o}{u}{t}{i}{n}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_SUBROUTINE );
}

{t}{h}{e}{n} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_THEN );
}

{t}{o} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_TO );
}

{w}{r}{i}{t}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_WRITE );
}

{u}{n}{d}{e}{f}{i}{n}{e}{d} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_UNDEFINED );
}

[%a-zA-Z][_a-zA-Z0-9]* {
#ifdef DEBUG
    ECHO;
#endif
    yylval = duplicate( uppercase( yytext ) );
    return( IDENTIFIER );
}

^[0-9][0-9][0-9][0-9][0-9][\ ] {
#ifdef DEBUG
    ECHO;
#endif
    yylval = duplicate( non_blank( yytext ) );
    return( LABEL );
}

[0-9]+ |
[0-9]+/\.[a-zA-Z]+\.\. {
#ifdef DEBUG
    ECHO;
#endif
    yylval = duplicate( yytext );
    return( INTEGER );
}

```

```

[0-9]+\.[0-9]*([eE][\+\-]?[0-9]+)? |
[0-9]*\.[0-9]+([eE][\+\-]?[0-9]+)? |
[0-9]+([eE][\+\-]?[0-9]+)? {
#ifdef DEBUG
    ECHO;
#endif
    yyval = duplicate( yytext );
    return( REAL );
}

[0-9]+\.[0-9]*([dD][\+\-]?[0-9]+)? |
[0-9]*\.[0-9]+([dD][\+\-]?[0-9]+)? |
[0-9]+([dD][\+\-]?[0-9]+)? {
#ifdef DEBUG
    ECHO;
#endif
    yyval = duplicate( yytext );
    return( DOUBLE_PRECISION );
}

\[^\']*\\' |
\[^\"]*\\" {
#ifdef DEBUG
    ECHO;
#endif
    yytext[ 0 ] = '\\';
    yytext[ strlen( yytext ) - 1 ] = '\\';
    yyval = duplicate( yytext );
    return( STRING );
}

[0-9]+[hH] {
#ifdef DEBUG
    ECHO;
#endif
    yyval = duplicate( hollerith( yytext, '\\'' ) );
    return( HOLLERITH );
}

```

FILE: declare/statement/Makefile

```

#
# Copyright 1991
# Georgia Institute of Technology
# Computer Engineering Research Laboratory
# Author: Stephen R. Wachtel
#

```

```

CC = cc -g
INCLUDE = ../include
CFLAGS = -IS(INCLUDE)
LIBRARY = statement.a

```

```

OBJECTS = \
    block_data_statement.o \
    common_statement.o \
    data_statement.o \
    declaration_statement.o \
    dimension_statement.o \
    do_statement.o \
    equivalence_statement.o \
    end_statement.o \
    function_statement.o \
    implicit_statement.o \
    parameter_statement.o \
    program_statement.o \
    subroutine_statement.o

```

```

$(LIBRARY): $(OBJECTS)
    rm -f $(LIBRARY)
    ar crv $(LIBRARY) $(OBJECTS)

```

```

ranlib $(LIBRARY)

.SUFFIXES: .c .o
.c.o:
    $(CC) -c $(CFLAGS) $<

clean:
    rm -f $(LIBRARY) $(OBJECTS)

FILE: declare/statement/block_data_statement.c

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include "list.h"
#include "table.h"
#include "attribute.h"

void block_data_statement( identifier )
register char *identifier;
{
    number_table = add_table( identifier, IMPLICIT | GLOBAL | VARIABLE | PROGRAM );
} /* block_data_statement */

FILE: declare/statement/common_statement.c

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include "list.h"
#include "table.h"
#include "attribute.h"

extern char *duplicate( );
extern char *parse( );

void common_statement( common_name, common_list )
register char *common_name;
register char *common_list;
{
    register char *common;
    register char *identifier;
    register char *subscript_list;
    register LIST *temporary;

    if ( common_name == 0 )
        common_name = duplicate( "BLKCOM" );
    add_list( 0, common_name, 0, 0, IMPLICIT | GLOBAL | CONSTANT | COMMON );

    if ( find_table( common_name ) == -1 )
        add_table( common_name, IMPLICIT | GLOBAL | CONSTANT | COMMON );

    while ( common = parse( common_list ) )
    {
        identifier = parse( common );
        subscript_list = parse( common );

        temporary = delete_list( 0, identifier );

        if ( subscript_list == 0 )
            add_list( common_name, identifier, 0, subscript_list, IMPLICIT | LOCAL |
VARIABLE );

```

```

        else
            add_list( common_name, identifier, 0, subscript_list, IMPLICIT | LOCAL |
VARIABLE | ARRAY );
        if ( temporary != 0 )
            add_list( common_name, identifier, temporary->type, temporary->subscript_list,
temporary->attribute );
    }
} /* common_statement */

```

FILE: declare/statement/data\_statement.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```

#include <string.h>
#include "list.h"
#include "attribute.h"

```

```

extern char *parse( );
extern char *list( );
extern char *merge( );

```

```

void data_statement( data_list )
register char *data_list;
{
    register char *data;
    register char *variable;
    register char *constant_list;
    register LIST *temporary;

    while ( data = parse( data_list ) )
    {
        variable = parse( data );
        variable[ strcspn( variable, "(" ) ] = '\0';

        constant_list = parse( data );
        constant_list = list( constant_list, "," );

        temporary = add_list( 0, variable, 0, 0, IMPLICIT | LOCAL | VARIABLE );
        temporary->number--;

        if ( temporary->data != 0 )
            temporary->data = merge( "%s,%s", temporary->data, constant_list );
        else
            temporary->data = constant_list;
    }
} /* data_statement */

```

FILE: declare/statement/declaration\_statement.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```

#include "list.h"
#include "attribute.h"

```

```

extern char *parse( );

```

```

void declaration_statement( type, declaration_list )
register char *type;
register char *declaration_list;
{

```

```

register char *declaration;
register char *identifier;
register char *subscript_list;

while ( declaration = parse( declaration_list ) )
{
    identifier = parse( declaration );
    subscript_list = parse( declaration );

    if ( subscript_list == 0 )
        add_list( 0, identifier, type, subscript_list, EXPLICIT | LOCAL | VARIABLE );
    else
        add_list( 0, identifier, type, subscript_list, EXPLICIT | LOCAL | VARIABLE |
ARRAY );
}
} /* declaration_statement */

```

FILE: declare/statement/dimension\_statement.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include "list.h"
#include "attribute.h"

extern char *parse( );

void dimension_statement( dimension_list )
register char *dimension_list;
{
    register char *dimension;
    register char *identifier;
    register char *subscript_list;

    while ( dimension = parse( dimension_list ) )
    {
        identifier = parse( dimension );
        subscript_list = parse( dimension );

        add_list( 0, identifier, 0, subscript_list, IMPLICIT | LOCAL | VARIABLE | ARRAY );
    }
} /* dimension_statement */

```

FILE: declare/statement/do\_statement.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include "list.h"
#include "attribute.h"

void do_statement( label, identifier, expression_list )
register char *label;
register char *identifier;
register char *expression_list;
{
    add_list( 0, identifier, 0, 0, IMPLICIT | LOCAL | VARIABLE )->number++;
} /* do_statement */

```

FILE: declare/statement/end\_statement.c

```

/*

```



```

* Copyright 1991
* Georgia Institute of Technology
* Computer Engineering Research Laboratory
* Author: Stephen R. Wachtel
*/

```

```

void end_statement( )
{
    implicit_initialize( );
} /* end_statement */

```

FILE: declare/statement/equivalence\_statement.c

```

/*
* Copyright 1991
* Georgia Institute of Technology
* Computer Engineering Research Laboratory
* Author: Stephen R. Wachtel
*/

#include "list.h"
#include "attribute.h"

extern char *parse( );

void equivalence_statement( equivalence_list )
register char *equivalence_list;
{
    register char *variable_list;
    register char *variable;
    register char *identifier;
    register char *subscript_list;

    while ( variable_list = parse( equivalence_list ) )
    {
        while ( variable = parse( variable_list ) )
        {
            identifier = parse( variable );
            subscript_list = parse( variable );

            add_list( 0, identifier, 0, 0, IMPLICIT | LOCAL | VARIABLE | EQUIVALENCE );
        }
    }
} /* equivalence_statement */

```

FILE: declare/statement/function\_statement.c

```

/*
* Copyright 1991
* Georgia Institute of Technology
* Computer Engineering Research Laboratory
* Author: Stephen R. Wachtel
*/

#include "list.h"
#include "table.h"
#include "attribute.h"

extern char *parse( );

void function_statement( optional_type, identifier, optional_formal_argument_list )
register char *optional_type;
register char *identifier;
register char *optional_formal_argument_list;
{
    register char *formal_argument;

    number_table = add_table( identifier, IMPLICIT | GLOBAL | VARIABLE | FUNCTION );

    if ( optional_type != 0 )

```

```

        add_list( 0, identifier, optional_type, 0, EXPLICIT | LOCAL | VARIABLE | FUNCTION
);
    else
        add_list( 0, identifier, optional_type, 0, IMPLICIT | LOCAL | VARIABLE | FUNCTION
);
    if ( optional_formal_argument_list != 0 )
    {
        while ( formal_argument = parse( optional_formal_argument_list ) )
            add_list( 0, formal_argument, 0, 0, IMPLICIT | LOCAL | VARIABLE |
FORMAL_ARGUMENT );
    }
} /* function_statement */

```

FILE: declare/statement/implicit\_statement.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

extern char *parse( );

void implicit_statement( type, implicit_list )
register char *type;
register char *implicit_list;
{
    register char *implicit;
    register char *lower_bound;
    register char *upper_bound;

    while ( implicit = parse( implicit_list ) )
    {
        lower_bound = parse( implicit );
        upper_bound = parse( implicit );

        type_implicit( type, lower_bound, upper_bound );
    }

    update_list( 0 );
} /* implicit_statement */

```

FILE: declare/statement/parameter\_statement.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include "list.h"
#include "attribute.h"

extern char *parse( );

void parameter_statement( parameter_list )
register char *parameter_list;
{
    register char *parameter;
    register char *identifier;
    register char *expression;
    register LIST *temporary;

    while ( parameter = parse( parameter_list ) )
    {
        identifier = parse( parameter );
        expression = parse( parameter );

        temporary = add_list( 0, identifier, 0, 0, IMPLICIT | LOCAL | CONSTANT );
    }
}

```

```

        temporary->number++;
        temporary->data = expression;
    }
} /* parameter_statement */

```

FILE: declare/statement/program\_statement.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```

#include "list.h"
#include "table.h"
#include "attribute.h"

```

```

void program_statement( identifier )
register char *identifier;
{
    number_table = add_table( identifier, IMPLICIT | GLOBAL | CONSTANT | PROGRAM );
} /* program_statement */

```

FILE: declare/statement/subroutine\_statement.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```

#include "list.h"
#include "table.h"
#include "attribute.h"

```

```

extern char *parse( );

```

```

void subroutine_statement( identifier, optional_formal_argument_list )
register char *identifier;
register char *optional_formal_argument_list;
{
    register char *formal_argument;

    number_table = add_table( identifier, IMPLICIT | GLOBAL | CONSTANT | FUNCTION );

    if ( optional_formal_argument_list != 0 )
    {
        while ( formal_argument = parse( optional_formal_argument_list ) )
            add_list( 0, formal_argument, 0, 0, IMPLICIT | LOCAL | VARIABLE |
FORMAL_ARGUMENT );
    }
} /* subroutine_statement */

```

## 12. Appendix G: equivalence program source

FILE: equivalence/Makefile

```
#
# Copyright 1991
# Georgia Institute of Technology
# Computer Engineering Research Laboratory
# Author: Stephen R. Wachtel
#

default:    equivalence

CC = cc -g
INCLUDE = include
CFLAGS = -I$(INCLUDE)
LIBRARY = statement/statement.a library/library.a

OBJECTS = \
$(INCLUDE)/grammar.h \
*grammar.[co] \
*scanner.[co] \
yytrace.[co] \
y.output

PROGRAMS = \
*equivalence

grammar.c: grammar.y
yacc -dv grammar.y
mv y.tab.h $(INCLUDE)/grammar.h
mv y.tab.c grammar.c

scanner.c: scanner.l
lex -vt scanner.l | sed 's/getc/yygetc/' >scanner.c

scanner.o: scanner.c
$(CC) $(CFLAGS) -c scanner.c

grammar.o: grammar.c
$(CC) $(CFLAGS) -c grammar.c

equivalence: grammar.o scanner.o $(LIBRARY)
$(CC) -o equivalence grammar.o scanner.o $(LIBRARY)

sgrammar.c: grammar.c yytoken.awk
awk -f yytoken.awk <grammar.c >sgrammar.c

sgrammar.o: sgrammar.c
$(CC) $(CFLAGS) -c sgrammar.c

sequivalence: sgrammar.o scanner.o $(LIBRARY)
$(CC) -o sequivalence sgrammar.o scanner.o $(LIBRARY)

dscanner.c: scanner.c
cp scanner.c dscanner.c

dscanner.o: dscanner.c $(INCLUDE)/grammar.h
$(CC) $(CFLAGS) -DDEBUG -c dscanner.c

dequivalence: grammar.o dscanner.o $(LIBRARY)
$(CC) -o dequivalence grammar.o dscanner.o $(LIBRARY)

tgrammar.c: grammar.c
sed 's/yystack:/* yytrace(yystate);/' <grammar.c >tgrammar.c

tgrammar.o: tgrammar.c
$(CC) $(CFLAGS) -c tgrammar.c
```

```
tequivalence: tgrammar.o scanner.o yytrace.o $(LIBRARY)
               $(CC) -o tequivalence tgrammar.o scanner.o yytrace.o $(LIBRARY)
```

```
yytrace.c: grammar.c yytrace.awk
           awk -f yytrace.awk <y.output >yytrace.c
```

```
yytrace.o: yytrace.c
           $(CC) $(CFLAGS) -c yytrace.c
```

```
clean:
        cd statement; make clean
        cd library; make clean
        rm -f $(PROGRAMS) $(OBJECTS)
```

```
FILE: equivalence/grammar.y
```

```
/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */
```

```
/*
 * FORTRAN 77
 */
```

```
%token RW_AND
%token RW_ASSIGN
%token RW_BACKSPACE
%token RW_BLOCK_DATA
%token RW_CALL
%token RW_CHARACTER
%token RW_CLOSE
%token RW_COMMON
%token RW_COMPLEX
%token RW_CONTINUE
%token RW_DATA
%token RW_DIMENSION
%token RW_DO
%token RW_DOUBLE_PRECISION
%token RW_ELSE
%token RW_ELSE_IF
%token RW_END
%token RW_END IF
%token RW_ENDFILE
%token RW_ENTRY
%token RW_EQ
%token RW_EQUIVALENCE
%token RW_EQV
%token RW_EXTERNAL
%token RW_FALSE
%token RW_FORMAT
%token RW_FUNCTION
%token RW_GE
%token RW_GO_TO
%token RW_GT
%token RW_IF
%token RW_IMPLICIT
%token RW_INCLUDE
%token RW_INQUIRE
%token RW_INTEGER
%token RW_INTRINSIC
%token RW_LE
%token RW_LOGICAL
%token RW_LT
%token RW_NAMELIST
%token RW_NE
%token RW_NEQV
%token RW_NOT
%token RW_OPEN
%token RW_OR
%token RW_PARAMETER
%token RW_PAUSE
%token RW_PRINT
```

```

%token RW_PROGRAM
%token RW_READ
%token RW_REAL
%token RW_RETURN
%token RW_REWIND
%token RW_SAVE
%token RW_STOP
%token RW_SUBROUTINE
%token RW_THEN
%token RW_TO
%token RW_TRUE
%token RW_WRITE
%token RW_UNDEFINED

%token COMMENT
%token CONCATENATE
%token DOUBLE_PRECISION
%token EXPONENTIATE
%token HOLLERITH
%token IDENTIFIER
%token INTEGER
%token LABEL
%token REAL
%token STRING

%left ','
%nonassoc ':'
%right '='
%left RW_EQV RW_NEQV
%left RW_OR
%left RW_AND
%left RW_NOT
%nonassoc RW_EQ RW_NE RW_LT RW_LE RW_GT RW_GE
%left CONCATENATE
%left '+' '-'
%left '*' '/'
%right EXPONENTIATE
%left SIGN

%{
typedef char *POINTER;
#define YYSTYPE POINTER

#include "list.h"
LIST* rlist = 0;
LIST* ilist = 0;

char *block_name = 0;
char *common_name = 0;

extern POINTER duplicate( );
extern POINTER merge( );
extern POINTER list( );
extern POINTER type( );
extern POINTER replicate( );
%}

%%

program:
    optional_statement_list
    ;

optional_statement_list:
    /* NULL */
    |
        statement_list
    ;

statement_list:
    statement
    |
        statement_list statement
    ;

```

```

statement:
    comment_statement
    |
    label unlabeled_statement
    ;

comment_statement:
    COMMENT
    ;

label:
    LABEL
    ;

unlabeled_statement:
    include_statement
    |
    program_statement
    |
    block_data_statement
    |
    function_statement
    |
    subroutine_statement
    |
    entry_statement
    |
    end_statement
    |
    specification_statement
    |
    executable_statement
    |
    format_statement
    ;

include_statement:
    RW_INCLUDE character_constant
    ;

program_statement:
    RW_PROGRAM program_identifier
    ;

program_identifier:
    IDENTIFIER
    {
        $$ = block_name = $1;
    }
    ;

block_data_statement:
    RW_BLOCK_DATA block_data_identifier
    ;

block_data_identifier:
    IDENTIFIER
    {
        $$ = block_name = $1;
    }
    ;

function_statement:
    RW_FUNCTION function_identifier optional_formal_argument_list
    |
    type RW_FUNCTION function_identifier optional_formal_argument_list
    ;

function_identifier:

```

```

IDENTIFIER
{
    $$ = block_name = $1;
}
;

subroutine_statement:
    RW_SUBROUTINE subroutine_identifier
    |
    RW_SUBROUTINE subroutine_identifier optional_formal_argument_list
;

subroutine_identifier:
    IDENTIFIER
    {
        $$ = block_name = $1;
    }
;

entry_statement:
    RW_ENTRY entry_identifier
    |
    RW_ENTRY entry_identifier optional_formal_argument_list
;

entry_identifier:
    IDENTIFIER
    {
        $$ = $1;
    }
;

optional_formal_argument_list:
    '(' ')'
    {
        $$ = 0;
    }
    |
    '(' formal_argument_list ')'
    {
        $$ = $2;
    }
;

formal_argument_list:
    formal_argument
    {
        $$ = merge( "%s", $1 );
    }
    |
    formal_argument_list ',' formal_argument
    {
        $$ = merge( "%s(%s)", $1, $3 );
    }
;

formal_argument:
    IDENTIFIER
    {
        $$ = $1;
    }
    |
    formal_argument_alternate_return
    {
        $$ = $1;
    }
;

formal_argument_alternate_return:
    '(' '*'
    {
        $$ = duplicate( "*" );
    }
;

```



```

;

end_statement:
    RW_END
    {
        print_list( rlist, block_name );
        delete_list( rlist );
        rlist = 0;

        print_list( ilist, block_name );
        delete_list( ilist );
        ilist = 0;

        block_name = 0;
    }
;

```

```

specification_statement:
    external_statement
    |
    intrinsic_statement
    |
    parameter_statement
    |
    dimension_statement
    |
    declaration_statement
    |
    save_statement
    |
    common_statement
    |
    equivalence_statement
    |
    implicit_statement
    |
    data_statement
    |
    namelist_statement
;

```

```

external_statement:
    RW_EXTERNAL external_list
;

```

```

external_list:
    external
    {
        $$ = merge( "{%s}", $1 );
    }
    |
    external_list ',' external
    {
        $$ = merge( "%s{%s}", $1, $3 );
    }
;

```

```

external:
    IDENTIFIER
    {
        $$ = $1;
    }
;

```

```

intrinsic_statement:
    RW_INTRINSIC intrinsic_list
;

```

```

intrinsic_list:
    intrinsic
    {
        $$ = merge( "{%s}", $1 );
    }
;

```

```

    intrinsic_list ',' intrinsic
    {
        $$ = merge( "%s{%s}", $1, $3 );
    }
;

intrinsic:
    IDENTIFIER
    {
        $$ = $1;
    }
;

parameter_statement:
    RW_PARAMETER '(' parameter_list ')'
;

parameter_list:
    parameter
    {
        $$ = merge( "{%s}", $1 );
    }
    |
    parameter_list ',' parameter
    {
        $$ = merge( "%s{%s}", $1, $3 );
    }
;

parameter:
    IDENTIFIER '=' expression
    {
        $$ = merge( "{%s}{%s}", $1, $3 );
    }
;

dimension_statement:
    RW_DIMENSION dimension_list
;

dimension_list:
    dimension
    {
        $$ = merge( "{%s}", $1 );
    }
    |
    dimension_list ',' dimension
    {
        $$ = merge( "%s{%s}", $1, $3 );
    }
;

dimension:
    IDENTIFIER '(' subscript_list ')'
    {
        $$ = merge( "{%s}{%s}", $1, $3 );
    }
;

subscript_list:
    subscript
    {
        $$ = merge( "{%s}", $1 );
    }
    |
    subscript_list ',' subscript
    {
        $$ = merge( "%s{%s}", $1, $3 );
    }
;

subscript:

```

```

upper_bound
{
    $$ = $1;
}
|
lower_bound ':' upper_bound
{
    $$ = merge( "%s:%s", $1, $3 );
}
;

lower_bound:
expression
{
    $$ = $1;
}
;

upper_bound:
expression
{
    $$ = $1;
}
|
upper_bound_adjustable
{
    $$ = $1;
}
;

upper_bound_adjustable:
"++"
{
    $$ = duplicate( "*" );
}
;

declaration_statement:
type declaration_list
;

declaration_list:
declaration
{
    $$ = merge( "{%s}", $1 );
}
|
declaration_list ',' declaration
{
    $$ = merge( "%s{%s}", $1, $3 );
}
;

declaration:
IDENTIFIER
{
    $$ = merge( "{%s}", $1 );
}
|
IDENTIFIER '(' subscript_list ')'
{
    $$ = merge( "{%s}{%s}", $1, $3 );
}
;

type:
type_name optional_type_length
{
    $$ = type( $1, $2 );
}
;

type_name:

```

```

RW_CHARACTER
{
    $$ = duplicate( "CHARACTER" );
}
|
RW_COMPLEX
{
    $$ = duplicate( "COMPLEX" );
}
|
RW_DOUBLE_PRECISION
{
    $$ = duplicate( "DOUBLE_PRECISION" );
}
|
RW_INTEGER
{
    $$ = duplicate( "INTEGER" );
}
|
RW_LOGICAL
{
    $$ = duplicate( "LOGICAL" );
}
|
RW_REAL
{
    $$ = duplicate( "REAL" );
}
|
RW_UNDEFINED
{
    $$ = duplicate( "UNDEFINED" );
}
;

optional_type_length:
/* NULL */
{
    $$ = 0;
}
|
type_length
{
    $$ = $1;
}
;

type_length:
'-' INTEGER
{
    $$ = $2;
}
|
'-' type_length_adjustable
{
    $$ = $2;
}
;

type_length_adjustable:
'(' '1' ')'
{
    $$ = duplicate( "-1" );
}
;

save_statement:
RW_SAVE optional_save_list
;

optional_save_list:
/* NULL */
{
    $$ = 0;
}

```

```

    |
    |   save_list
    |   {
    |       $$ = $1;
    |   }
    |
;

save_list:
    save
    {
        $$ = merge( "{%s}", $1 );
    }
    |
    |   save_list ',' save
    |   {
    |       $$ = merge( "%s{%s}", $1, $3 );
    |   }
    |
;

save:
    IDENTIFIER
    {
        $$ = $1;
    }
    |
    |   common_name
    |   {
    |       $$ = $1;
    |   }
    |
;

common_statement:
    RW_COMMON optional_common_name common_variable_list
    {
        common_name = 0;
    }
    |
;

optional_common_name:
    /* NULL */
    {
        $$ = common_name = 0;
    }
    |
    |   common_name
    |   {
    |       $$ = common_name = $1;
    |   }
    |
;

common_name:
    '/' optional_identifier '/'
    {
        $$ = $2;
    }
    |
;

optional_identifier:
    /* NULL */
    {
        $$ = 0;
    }
    |
    |   IDENTIFIER
    |   {
    |       $$ = $1;
    |   }
    |
;

common_variable_list:
    common_variable
    {
        $$ = merge( "{%s}", $1 );
    }
    |
;

```

```

    common_variable_list ',' common_variable
    {
        $$ = merge( "%s%s", $1, $3 );
    }
;

common_variable:
IDENTIFIER
{
    $$ = merge( "{%s}", $1 );
}
|
IDENTIFIER '(' subscript_list ')'
{
    $$ = merge( "%s{%s}", $1, $3 );
}
;

equivalence_statement:
RW_EQUIVALENCE equivalence_list
;

equivalence_list:
equivalence
{
    $$ = merge( "{%s}", $1 );
}
|
equivalence_list ',' equivalence
{
    $$ = merge( "%s%s", $1, $3 );
}
;

equivalence:
 '(' equivalence_variable ',' equivalence_variable ')'
{
    if ( strcmp( $2, "VAR", 3 ) == 0 )
        add_list( &rlist, 0, $4 );

    if ( strcmp( $2, "IVAR", 4 ) == 0 )
        add_list( &ilist, 0, $4 );

    $$ = merge( "%s{%s}", $2, $4 );
}
;

equivalence_variable:
IDENTIFIER
{
    $$ = $1;
}
|
IDENTIFIER '(' subscript_list ')'
{
    $$ = $1;
}
;

implicit_statement:
RW_IMPLICIT type '(' implicit_list ')'
;

implicit_list:
implicit
{
    $$ = merge( "{%s}", $1 );
}
|
implicit_list ',' implicit
{
    $$ = merge( "%s%s", $1, $3 );
}

```

```

;

implicit:
  IDENTIFIER
  {
    $$ = merge( "{%s}", $1 );
  }
  |
  IDENTIFIER '-' IDENTIFIER
  {
    $$ = merge( "{%s}{%s}", $1, $3 );
  }
;

namelist_statement:
  RW_NAMELIST namelist_name namelist_list
;

namelist_name:
  '/' IDENTIFIER '/'
  {
    $$ = $2;
  }
;

namelist_list:
  namelist
  {
    $$ = merge( "{%s}", $1 );
  }
  |
  namelist_list ',' namelist
  {
    $$ = merge( "%s{%s}", $1, $3 );
  }
;

namelist:
  IDENTIFIER
  {
    $$ = $1;
  }
;

data_statement:
  RW_DATA data_list
  {
    data_statement( $2 );
  }
;

data_list:
  data
  {
    $$ = merge( "{%s}", $1 );
  }
  |
  data_list optional_comma data
  {
    $$ = merge( "%s{%s}", $1, $3 );
  }
;

data:
  data_variable_list '/' data_constant_list '/'
  {
    $$ = merge( "{%s}{%s}", $1, $3 );
  }
;

data_variable_list:
  data_variable

```

```

    {
        $$ = merge( "{%s}", $1 );
    }
|
data_variable_list ',' data_variable
{
    $$ = merge( "%s{%s}", $1, $3 );
}
;

data_variable:
    variable
    {
        $$ = $1;
    }
|
    data_implied_do_list
    {
        $$ = $1;
    }
;

data_implied_do_list:
    '(' data_variable_list ',' IDENTIFIER '=' expression_list ')'
    {
        $$ = merge( "( %s, %s = %s )", list( $2, " , " ), $4, list( $6, " , " ) );
    }
;

data_constant_list:
    data_constant
    {
        $$ = merge( "{%s}", $1 );
    }
|
    data_constant_list ',' data_constant
    {
        $$ = merge( "%s{%s}", $1, $3 );
    }
;

data_constant:
    data_initialization
    {
        $$ = $1;
    }
|
    IDENTIFIER '*' data_initialization
    {
        $$ = merge( "%s * %s", $1, $3 );
    }
|
    INTEGER '*' data_initialization
    {
        $$ = replicate( atoi( $1 ), $3, "{}{ " );
    }
;

data_initialization:
    IDENTIFIER
    {
        $$ = $1;
    }
|
    character_constant
    {
        $$ = $1;
    }
|
    logical_constant
    {
        $$ = $1;
    }
|
    signed_numerical_constant
    {

```



```

        $$ = $1;
    }
;

signed_numerical_constant:
    numerical_constant
    {
        $$ = $1;
    }
|
    '+' numerical_constant %prec SIGN
    {
        $$ = merge( "+%s", $2 );
    }
|
    '-' numerical_constant %prec SIGN
    {
        $$ = merge( "-%s", $2 );
    }
;

expression:
    parenthesis_expression
    {
        $$ = $1;
    }
|
    simple_expression
    {
        $$ = $1;
    }
;

parenthesis_expression:
    '(' expression ')'
    {
        $$ = merge( "( %s )", $2 );
    }
;

simple_expression:
    variable
    {
        $$ = $1;
    }
|
    constant
    {
        $$ = $1;
    }
|
    arithmetic_expression
    {
        $$ = $1;
    }
|
    character_expression
    {
        $$ = $1;
    }
|
    relational_expression
    {
        $$ = $1;
    }
|
    logical_expression
    {
        $$ = $1;
    }
|
    unary_expression
    {
        $$ = $1;
    }
;

```

```

variable:
  IDENTIFIER
  {
    usage_list( rlist, $1 );
    usage_list( ilist, $1 );

    $$ = $1;
  }
  |
  IDENTIFIER string_subset
  {
    usage_list( rlist, $1 );
    usage_list( ilist, $1 );

    $$ = merge( "%s%s", $1, $2 );
  }
  |
  array
  {
    $$ = $1;
  }
;

array:
  IDENTIFIER '(' optional_expression_list ')'
  {
    usage_list( rlist, $1 );
    usage_list( ilist, $1 );

    $$ = merge( "%s(%s)", $1, list( $3, ", " ) );
  }
  |
  IDENTIFIER '(' optional_expression_list ')' string_subset
  {
    usage_list( rlist, $1 );
    usage_list( ilist, $1 );

    $$ = merge( "%s(%s)%s", $1, list( $3, ", " ), $5 );
  }
;

optional_expression_list:
  /* NULL */
  {
    $$ = 0;
  }
  |
  expression_list
  {
    $$ = $1;
  }
;

expression_list:
  expression
  {
    $$ = merge( "{%s}", $1 );
  }
  |
  expression_list ',' expression
  {
    $$ = merge( "%s{%s}", $1, $3 );
  }
;

string_subset:
  '(' optional_expression ':' optional_expression ')'
  {
    $$ = merge( "( %s : %s )", $2, $4 );
  }
;

optional_expression:
  /* NULL */
  {

```

```

        $$ = 0;
    }
    {
        expression
        {
            $$ = $1;
        }
    }
;

constant:
    character_constant
    {
        $$ = $1;
    }
    |
    logical_constant
    {
        $$ = $1;
    }
    |
    numerical_constant
    {
        $$ = $1;
    }
;

character_constant:
    HOLLERITH
    {
        $$ = $1;
    }
    |
    STRING
    {
        $$ = $1;
    }
;

logical_constant:
    RW_FALSE
    {
        $$ = duplicate( ".FALSE." );
    }
    |
    RW_TRUE
    {
        $$ = duplicate( ".TRUE." );
    }
;

numerical_constant:
    DOUBLE_PRECISION
    {
        $$ = $1;
    }
    |
    INTEGER
    {
        $$ = $1;
    }
    |
    REAL
    {
        $$ = $1;
    }
;

arithmetic_expression:
    expression '+' expression %prec '+'
    {
        $$ = merge( "%s + %s", $1, $3 );
    }
    |
    expression '-' expression %prec '-'
    {
        $$ = merge( "%s - %s", $1, $3 );
    }
;

```

```

    }
    expression '*' expression %prec '*'
    {
        $$ = merge( "%s * %s", $1, $3 );
    }
    expression '/' expression %prec '/'
    {
        $$ = merge( "%s / %s", $1, $3 );
    }
    expression EXPONENTIATE expression %prec EXPONENTIATE
    {
        $$ = merge( "%s ** %s", $1, $3 );
    }
;

```

```

character_expression:
    expression '/' '/' expression %prec CONCATENATE
    {
        $$ = merge( "%s // %s", $1, $4 );
    }
;

```

```

relational_expression:
    expression RW_EQ expression %prec RW_EQ
    {
        $$ = merge( "%s .EQ. %s", $1, $3 );
    }
    expression RW_NE expression %prec RW_NE
    {
        $$ = merge( "%s .NE. %s", $1, $3 );
    }
    expression RW_LT expression %prec RW_LT
    {
        $$ = merge( "%s .LT. %s", $1, $3 );
    }
    expression RW_LE expression %prec RW_LE
    {
        $$ = merge( "%s .LE. %s", $1, $3 );
    }
    expression RW_GT expression %prec RW_GT
    {
        $$ = merge( "%s .GT. %s", $1, $3 );
    }
    expression RW_GE expression %prec RW_GE
    {
        $$ = merge( "%s .GE. %s", $1, $3 );
    }
;

```

```

logical_expression:
    expression RW_AND expression %prec RW_AND
    {
        $$ = merge( "%s .AND. %s", $1, $3 );
    }
    expression RW_OR expression %prec RW_OR
    {
        $$ = merge( "%s .OR. %s", $1, $3 );
    }
    expression RW_EQV expression %prec RW_EQV
    {
        $$ = merge( "%s .EQV. %s", $1, $3 );
    }
    expression RW_NEQV expression %prec RW_NEQV
    {
        $$ = merge( "%s .NEQV. %s", $1, $3 );
    }
;

```

```

unary_expression:
    '+' expression %prec SIGN
    {
        $$ = merge( "+%s", $2 );
    }
    |
    '-' expression %prec SIGN
    {
        $$ = merge( "-%s", $2 );
    }
    |
    RW_NOT expression %prec RW_NOT
    {
        $$ = merge( ".NOT. %s", $2 );
    }
    ;

executable_statement:
    do_statement
    |
    logical_if_statement
    |
    block_if_statement
    |
    else_statement
    |
    else_if_statement
    |
    end_if_statement
    |
    subset_executable_statement
    ;

do_statement:
    RW_DO optional_integer IDENTIFIER '=' expression_list
    ;

optional_integer:
    /* NULL */
    {
        $$ = 0;
    }
    |
    INTEGER
    {
        $$ = $1;
    }
    ;

logical_if_statement:
    if_expression subset_executable_statement
    ;

if_expression:
    RW_IF '(' expression ')'
    ;

block_if_statement:
    RW_IF '(' expression ')' RW_THEN
    ;

else_statement:
    RW_ELSE
    ;

else_if_statement:
    RW_ELSE_IF '(' expression ')' RW_THEN
    ;

end_if_statement:

```

```

        RW_END_IF
    ;

subset_executable_statement:
    assignment_statement
    |
    assign_statement
    |
    arithmetic_if_statement
    |
    continue_statement
    |
    call_statement
    |
    return_statement
    |
    unconditional_go_to_statement
    |
    computed_go_to_statement
    |
    assigned_go_to_statement
    |
    stop_statement
    |
    pause_statement
    |
    io_statement
    ;

assignment_statement:
    variable '=' expression
    ;

assign_statement:
    RW_ASSIGN INTEGER RW_TO IDENTIFIER
    ;

arithmetic_if_statement:
    RW_IF '(' expression ')' integer_list
    ;

continue_statement:
    RW_CONTINUE
    ;

call_statement:
    RW_CALL IDENTIFIER
    |
    RW_CALL IDENTIFIER optional_actual_argument_list
    ;

optional_actual_argument_list:
    '(' ' '
    {
        $$ = 0;
    }
    |
    '(' actual_argument_list ')'
    {
        $$ = $2;
    }
    ;

actual_argument_list:
    actual_argument
    {
        $$ = merge( "%s", $1 );
    }
    |
    actual_argument_list ',' actual_argument
    {
        $$ = merge( "%s%s", $1, $3 );
    }
    ;

```

```

;

actual_argument:
    expression
    {
        $$ = $1;
    }
    |
    actual_argument_alternate_return
    {
        $$ = $1;
    }
;

actual_argument_alternate_return:
    '*' INTEGER
    {
        $$ = merge( "%s", $2 );
    }
;

return_statement:
    RW_RETURN optional_expression
;

unconditional_go_to_statement:
    RW_GO_TO INTEGER
;

computed_go_to_statement:
    RW_GO_TO '(' integer_list ')' optional_comma expression
;

assigned_go_to_statement:
    RW_GO_TO IDENTIFIER
    |
    RW_GO_TO IDENTIFIER optional_comma '(' integer_list ')'
;

optional_comma:
    /* NULL */
    |
    ','
;

integer_list:
    INTEGER
    {
        $$ = merge( "%s", $1 );
    }
    |
    integer_list ',' INTEGER
    {
        $$ = merge( "%s%s", $1, $3 );
    }
;

pause_statement:
    RW_PAUSE optional_expression
;

stop_statement:
    RW_STOP optional_expression
;

io_statement:
    open_statement
    |
    close_statement

```

```

    inquire_statement
  |
    read_statement
  |
    write_statement
  |
    print_statement
  |
    backspace_statement
  |
    rewind_statement
  |
    endfile_statement
;

open_statement:
  RW_OPEN '(' control_information_list ')'
;

close_statement:
  RW_CLOSE '(' control_information_list ')'
;

inquire_statement:
  RW_INQUIRE '(' control_information_list ')'
;

read_statement:
  RW_READ '(' control_information_list ')' optional_io_list
  |
  RW_READ control
  |
  RW_READ control ',' io_list
;

write_statement:
  RW_WRITE '(' control_information_list ')' optional_io_list
;

print_statement:
  RW_PRINT control
  |
  RW_PRINT control ',' io_list
;

backspace_statement:
  RW_BACKSPACE '(' control_information_list ')'
  |
  RW_BACKSPACE control
;

rewind_statement:
  RW_REWIND '(' control_information_list ')'
  |
  RW_REWIND control
;

endfile_statement:
  RW_ENDFILE '(' control_information_list ')'
  |
  RW_ENDFILE control
;

control_information_list:
  control_information
  {
    $$ = merge( "%s", $1 );
  }
  |
  control_information_list ',' control_information
  {

```



```

    $$ = merge( "%s{%s}", $1, $3 );
}
;

control_information:
control
{
    $$ = $1;
}
|
IDENTIFIER '=' control
{
    $$ = merge( "%s = %s", $1, $3 );
}
;

control:
variable
{
    $$ = $1;
}
|
constant
{
    $$ = $1;
}
|
'*'
{
    $$ = duplicate( "*" );
}
;

optional_io_list:
/* NULL */
{
    $$ = 0;
}
|
io_list
{
    $$ = $1;
}
;

io_list:
io
{
    $$ = merge( "%s)", $1 );
}
|
io_list ',' io
{
    $$ = merge( "%s{%s}", $1, $3 );
}
;

io:
expression
{
    $$ = $1;
}
|
io_implied_do_list
{
    $$ = $1;
}
;

io_implied_do_list:
'(' io_list '.' IDENTIFIER '=' expression_list ')'
{
    $$ = merge( "( %s, %s = %s )", list( $2, ", " ), $4, list( $6, ", " ) );
}
;

```

```
format_statement:
    RW_FORMAT
    ;
```

```
%%
```

```
FILE: equivalence/include/list.h
```

```
/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */
```

```
#define LIST struct list_type
LIST
{
    char *identifier;
    char *alternate;
    int usage;
    LIST *next;
};
```

```
extern LIST *end_list( );
extern LIST *add_list( );
extern LIST *find_list( );
extern void usage_list( );
extern LIST *find_index( );
extern void print_list( );
extern void delete_list( );
```

```
FILE: equivalence/library/Makefile
```

```
#
# Copyright 1991
# Georgia Institute of Technology
# Computer Engineering Research Laboratory
# Author: Stephen R. Wachtel
#
```

```
CC = cc -g
INCLUDE = ../include
CFLAGS = -I$(INCLUDE)
LIBRARY = library.a
```

```
OBJECTS = \
    count.o \
    duplicate.o \
    hollerith.o \
    link_list.o \
    list.o \
    main.o \
    merge.o \
    non_blank.o \
    parse.o \
    replicate.o \
    type.o \
    uppercase.o \
    yyerror.o \
    yygetc.o \
    yywrap.o
```

```
$(LIBRARY): $(OBJECTS)
    ar crv $(LIBRARY) $(OBJECTS)
    ranlib $(LIBRARY)
```

```
.SUFFIXES: .c .o
```

```
.c.o:
    $(CC) -c $(CFLAGS) $<
```

```
clean:
    rm -f $(LIBRARY) $(OBJECTS)
```

```
FILE: equivalence/library/count.c
```

```
/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */
```

```
int count( string, length, c )
register char *string;
register int length;
register char c;
{
    register int c_count = 0;

    while ( length != 0 )
    {
        if ( *string == c )
            c_count++;

        string++;
        length--;
    }

    return( c_count );
} /* count */
```

```
FILE: equivalence/library/duplicate.c
```

```
/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */
```

```
#include <stdio.h>
#include <string.h>
#include <malloc.h>
```

```
char *duplicate( string )
register char *string;
{
    register char *temporary = (char *)NULL;

    if ( string != (char *)NULL )
    {
        if ( ( temporary = (char *)malloc( strlen( string ) + 1 ) ) != (char *)NULL )
            strcpy( temporary, string );
        else
            fprintf( stderr, "ERROR: duplicate( %s )\n", string );
    }

    return( temporary );
} /* duplicate */
```

```
FILE: equivalence/library/hollerith.c
```

```
/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */
```

```
#include <stdio.h>

char *hollerith( string, delimiter )
register char *string;
register char delimiter;
{
    int hollerith_length;
    register int string_length = 0;

    sscanf( string, "%dh", &hollerith_length );

    string[ string_length++ ] = delimiter;
    while ( hollerith_length != 0 )
    {
        if ( ( string[ string_length ] = yyinput( ) ) == '\n' )
        {
            yyunput( string[ string_length ] );
            break;
        }

        string_length++;
        hollerith_length--;
    }
    string[ string_length++ ] = delimiter;

    string[ string_length ] = '\0';
    return( string );
} /* hollerith */
```

FILE: equivalence/library/link\_list.c

```
/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */
```

```
#include <stdio.h>
#include <string.h>
#include <malloc.h>
#include "list.h"
```

```
extern FILE *yyin;
extern FILE *yyout;
extern char *merge( );
```

```
LIST *end_list( list )
register LIST *list;
{
    if ( list != (LIST *)NULL )
    {
        while ( list->next != (LIST *)NULL )
            list = list->next;
    }

    return( list );
} /* end_list */
```

```
LIST *add_list( list, common_name, identifier )
register LIST **list;
register char *common_name;
register char *identifier;
{
    register LIST *temporary = (LIST *)malloc( sizeof( LIST ) );

    if ( common_name == (char *)NULL )
    {
        temporary->identifier = identifier;
        temporary->alternate = (char *)"NULL";
    }
    else
```

```

    {
        temporary->identifier = identifier;
        temporary->alternate = merge( "%s.%s", common_name, identifier );
    }

    temporary->usage = 0;
    temporary->next = (LIST *)NULL;

    if ( *list == (LIST *)NULL )
        *list = temporary;
    else
        end_list( *list )->next = temporary;

    return( temporary );
} /* add_list */

LIST *find_list( list, identifier )
register LIST *list;
register char *identifier;
{
    while ( list != (LIST *)NULL )
    {
        if ( strcmp( list->identifier, identifier ) == 0 )
            return( list );

        list = list->next;
    }

    return( (LIST *)NULL );
} /* find_list */

void usage_list( list, identifier )
register LIST *list;
register char *identifier;
{
    register LIST *temporary;

    if ( ( temporary = find_list( list, identifier ) ) != (LIST *)NULL )
        temporary->usage++;
} /* usage_list */

LIST *find_index( list, index )
register LIST *list;
register int index;
{
    while ( list != (LIST *)NULL )
    {
        if ( --index == 0 )
            return( list );

        list = list->next;
    }

    return( (LIST *)NULL );
} /* find_index */

void print_list( list, name )
register LIST *list;
register char *name;
{
    while ( list != (LIST *)NULL )
    {
        fprintf( yyout, "d(\\"UU%s.FOR\\",\\"%s\\",\\"%s\\",%d)\n", name, list->identifier,
list->alternate, list->usage );

        list = list->next;
    }
} /* print_list */

void delete_list( list )
register LIST *list;
{
    if ( list != (LIST *)NULL )
    {
        delete_list( list->next );
    }
}

```

```

        free( list );
    }
} /* delete_list */

FILE: equivalence/library/list.c

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wacntel
 */

extern char *parse( );
extern char *merge( );

char *list( input_list, delimiter )
register char *input_list;
register char *delimiter;
{
    register char *output_list;
    register char *list;
    register char *temporary;

    output_list = parse( input_list );
    list = parse( input_list );

    while ( list != (char *)0 )
    {
        temporary = merge( "%s%s%s", output_list, delimiter, list );

        free( output_list );
        free( list );

        output_list = temporary;
        list = parse( input_list );
    }

    return( output_list );
} /* list */

```

FILE: equivalence/library/main.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include <stdio.h>

extern FILE *yyin;
extern FILE *yyout;

#define PROGRAM argument[ 0 ]
#define INPUT_FILE argument[ 1 ]
#define OUTPUT_FILE argument[ 2 ]

int main( number_argument, argument )
int number_argument;
char *argument[ ];
{
    if ( number_argument == 1 )
    {
        yyin = stdin;
        yyout = stdout;

        yyparse( );
        exit( 0 );
    }
}

```

```

    if ( number_argument == 3 )
    {
        if ( ( yyin = fopen( INPUT_FILE, "r" ) ) == (FILE *)NULL )
        {
            fprintf( stderr, "%s: ERROR - unable to open input file '%s'\n", PROGRAM,
INPUT_FILE );
            exit( -1 );
        }

        if ( ( yyout = fopen( OUTPUT_FILE, "w" ) ) == (FILE *)NULL )
        {
            fprintf( stderr, "%s: ERROR - unable to open output file '%s'\n", PROGRAM,
OUTPUT_FILE );
            exit( -1 );
        }

        yyparse();
        exit( 0 );
    }

    fprintf( stderr, "usage: %s <input file> <output file>\n", PROGRAM );
    exit( 0 );
} /* main */

```

FILE: equivalence/library/merge.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include <stdio.h>
#include <string.h>
#include <malloc.h>

#define STRLEN( s ) ( strlen( s ) - 2 )

char *merge( string, a, b, c, d )
register char *string;
register char *a;
register char *b;
register char *c;
register char *d;
{
    register char *temporary = (char *)NULL;

    switch ( count( string, strlen( string ), '%' ) )
    {
        case 0:
            if ( ( temporary = (char *)malloc( strlen( string ) + 1 ) ) != (char *)NULL )
                sprintf( temporary, string );
            else
                fprintf( stderr, "ERROR: merge( %s )\n", string );
            break;

        case 1:
            if ( ( temporary = (char *)malloc( strlen( string ) + STRLEN( a ) + 1 ) ) !=
(char *)NULL )
                sprintf( temporary, string, a );
            else
                fprintf( stderr, "ERROR: merge( %s, %s )\n", string, a );
            break;

        case 2:
            if ( ( temporary = (char *)malloc( strlen( string ) + STRLEN( a ) + STRLEN( b
) + 1 ) ) != (char *)NULL )
                sprintf( temporary, string, a, b );
            else
                fprintf( stderr, "ERROR: merge( %s, %s, %s )\n", string, a, b );
            break;

        case 3:
            if ( ( temporary = (char *)malloc( strlen( string ) + STRLEN( a ) + STRLEN( b

```

```

) + STRLEN( c ) + 1 ) ) != (char *)NULL )
    sprintf( temporary, string, a, b, c );
else
    fprintf( stderr, "ERROR: merge( %s, %s, %s, %s )\n", string, a, b, c );
    break;

case 4:
    if ( ( temporary = (char *)malloc( strlen( string ) + STRLEN( a ) + STRLEN( b
) + STRLEN( c ) + STRLEN( d ) + 1 ) ) != (char *)NULL )
        sprintf( temporary, string, a, b, c, d );
    else
        fprintf( stderr, "ERROR: merge( %s, %s, %s, %s, %s )\n", string, a, b, c, d
);
    break;

default:
    fprintf( stderr, "ERROR: merge( %s )\n", string );
    break;
}

return( temporary );
} /* merge */

```

FILE: equivalence/library/non\_blank.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```
#include <string.h>
```

```

char *non_blank( string )
register Char *string;
{
    register int offset;
    register int length;

    length = strlen( string ) - 1;
    while ( ( string[ length ] == ' ' ) && ( string[ length ] != '\0' ) )
        string[ length-- ] = '\0';

    offset = 0;
    while ( ( string[ offset ] == ' ' ) && ( string[ offset ] != '\0' ) )
        string[ offset++ ] = '\0';

    strcpy( string, &string[ offset ] );

    if ( strlen( string ) != 0 )
        return( string );
    else
        return( 0 );
} /* non_blank */

```

FILE: equivalence/library/parse.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```
#include <string.h>
```

```
extern char *duplicate( );
```

```

char *parse( list )
register char *list;
{

```



```

register int length = 0;
register int brace = 0;
register char *temporary = (char *)0;

for (;;)
{
    switch ( list[ length ] )
    {
        case '[':
            brace++;
            break;

        case ']':
            brace--;
            break;

        }

    if ( brace == 0 )
        break;

    length++;

}

if ( length .
{
    list[ length ] = '\0';
    temporary = duplicate( list + 1 );
    strcpy( list, list + 1 + length );
}
else
{
    if ( list[ length ] != '\0' )
    {
        temporary = duplicate( list );
        list[ length ] = '\0';
    }
}

return( temporary );
} /* parse */

```

FILE: equivalence/library/replicate.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```

#include <string.h>
#include <malloc.h>

```

```

char *replicate( count, string, delimiter )
register int count;
register char *string;
register char *delimiter;
{
    register char *temporary = (char *)malloc( ( count * strlen( string ) ) + ( ( count -
1 ) * strlen( delimiter ) ) + 1 );

    if ( temporary != (char *)0 )
    {
        strcpy( temporary, string );

        while ( --count != 0 )
        {
            strcat( temporary, delimiter );
            strcat( temporary, string );
        }

        return( temporary );
    }
} /* replicate */

```

FILE: equivalence/library/type.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

extern char *merge( );

char *type( type_name, optional_type_length )
register char *type_name;
register char *optional_type_length;
{
    if ( optional_type_length != 0 )
        return( merge( "%s%s", type_name, optional_type_length ) );
    else
        return( type_name );
} /* type */

```

FILE: equivalence/library/uppercase.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

char *uppercase( string )
register char *string;
{
    register int index = 0;

    while ( string[ index ] != '\0' )
    {
        string[ index ] = toupper( string[ index ] );
        index++;
    }

    return( string );
} /* uppercase */

```

FILE: equivalence/library/yyerror.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include <stdio.h>

extern int yylineno;

void yyerror( string )
register char *string;
{
    fprintf( stderr, "line %d, %s\n", yylineno, string );

    exit( -1 );
} /* yyerror */

```

FILE: equivalence/library/yygetc.c

```

/*
 * Copyright 1991

```

```

* Georgia Institute of Technology
* Computer Engineering Research Laboratory
* Author: Stephen R. Wachtel
*/

```

```

#include <stdio.h>
#include <ctype.h>

```

```
extern int yylineno;
```

```

int tab( length )
register int length;
{
    while ( length-- != 0 )
        yyunput( ' ' );

    return( ' ' );
} /* tab */

```

```

int yygetc( file )
register FILE *file;
{
    int c;
    int column[ 6 ];

loop:
    if ( ( c = getc( file ) ) == '\t' )
        c = tab( 6 );

    if ( c != '\n' )
        return( c );

    if ( ( column[ 0 ] = getc( file ) ) != ' ' )
        goto abort_0;
    if ( ( column[ 1 ] = getc( file ) ) != ' ' )
        goto abort_1;
    if ( ( column[ 2 ] = getc( file ) ) != ' ' )
        goto abort_2;
    if ( ( column[ 3 ] = getc( file ) ) != ' ' )
        goto abort_3;
    if ( ( column[ 4 ] = getc( file ) ) != ' ' )
        goto abort_4;
    if ( isspace( column[ 5 ] = getc( file ) ) )
        goto abort_5;

    yylineno++;
    goto loop;

abort_5:
    if ( column[ 5 ] == '\t' )
        tab( 1 );
    else
    {
        yyunput( column[ 5 ] );
        if ( column[ 5 ] == '\n' )
            yylineno++;
    }

abort_4:
    if ( column[ 4 ] == '\t' )
        tab( 2 );
    else
    {
        yyunput( column[ 4 ] );
        if ( column[ 4 ] == '\n' )
            yylineno++;
    }

abort_3:
    if ( column[ 3 ] == '\t' )
        tab( 3 );
    else
    {
        yyunput( column[ 3 ] );
        if ( column[ 3 ] == '\n' )
            yylineno++;
    }
}

```

```

abort_2:
    if ( column[ 2 ] == '\t' )
        tab( 4 );
    else
    {
        yyunput( column[ 2 ] );
        if ( column[ 2 ] == '\n' )
            yylineno++;
    }

abort_1:
    if ( column[ 1 ] == '\t' )
        tab( 5 );
    else
    {
        yyunput( column[ 1 ] );
        if ( column[ 1 ] == '\n' )
            yylineno++;
    }

abort_0:
    if ( column[ 0 ] == '\t' )
        tab( 6 );
    else
    {
        yyunput( column[ 0 ] );
        if ( column[ 0 ] == '\n' )
            yylineno++;
    }

    return( c );
} /* yygetc */

```

FILE: equivalence/library/yywrap.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

int yywrap( )
{
    return( 1 );
} /* yywrap */

```

FILE: equivalence/scanner.l

```

%{
/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */
%}

%a 10000
%e 10000
%k 10000
%n 10000
%o 10000
%p 10000

a  [aA]
b  [bB]
c  [cC]
d  [dD]
e  [eE]
f  [fF]
g  [gG]
h  [hH]

```

```

i  [iI]
j  [jJ]
k  [kK]
l  [lL]
m  [mM]
n  [nN]
o  [oO]
p  [pP]
q  [qQ]
r  [rR]
s  [sS]
t  [tT]
u  [uU]
v  [vV]
w  [wW]
x  [xX]
y  [yY]
z  [zZ]

```

```

%(
#include "grammar.h"
extern char *yylval;

#undef YYLMAX
#define YYLMAX (256*20)

extern char *duplicate( );
extern char *hollerith( );
extern char *non_blank( );
extern char *uppercase( );
%)

%%

^[\\*cC].*[\\n]  |
^[\\ ]*[\\n] {
#ifdef DEBUG
    ECHO;
#endif
    yyval = duplicate( yytext );
    return( COMMENT );
}

[\\ ] {
#ifdef DEBUG
    ECHO;
#endif
    /* return( '\\ ' ) */;
}

[\\&] {
#ifdef DEBUG
    ECHO;
#endif
    return( '\\&' );
}

[\\(] {
#ifdef DEBUG
    ECHO;
#endif
    return( '\\(' );
}

[\\)] {
#ifdef DEBUG
    ECHO;
#endif
    return( '\\)' );
}

```

```
{\*} {
#ifdef DEBUG
    ECHO;
#endif
    return( '\*' );
}

{\[*\][\*]} {
#ifdef DEBUG
    ECHO;
#endif
    return( EXPONENTIATE );
}

{\\+} {
#ifdef DEBUG
    ECHO;
#endif
    return( '\\+' );
}

{\\,} {
#ifdef DEBUG
    ECHO;
#endif
    return( '\\,' );
}

{\\-} {
#ifdef DEBUG
    ECHO;
#endif
    return( '\\-' );
}

{\\.} {
#ifdef DEBUG
    ECHO;
#endif
    return( '\\.' );
}

{\\/} {
#ifdef DEBUG
    ECHO;
#endif
    return( '\\/' );
}

{\\:} {
#ifdef DEBUG
    ECHO;
#endif
    return( '\\:' );
}

{\\=} {
#ifdef DEBUG
    ECHO;
#endif
    return( '\\=' );
}

{\\n} {
#ifdef DEBUG
    ECHO;
#endif
    /* return( '\\n' ) */;
}

{\\t} {
```

```

#ifdef DEBUG
    ECHO;
#endif
    /* return( '\t' ) */;
}

```

```

[\.]{a}{n}{d}{\.} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_AND );
}

```

```

[\.]{e}{q}{\.} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_EQ );
}

```

```

[\.]{e}{q}{v}{\.} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_EQV );
}

```

```

[\.]{f}{a}{l}{s}{e}{\.} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_FALSE );
}

```

```

[\.]{g}{e}{\.} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_GE );
}

```

```

[\.]{g}{t}{\.} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_GT );
}

```

```

[\.]{l}{e}{\.} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_LE );
}

```

```

[\.]{l}{t}{\.} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_LT );
}

```

```

[\.]{n}{e}{\.} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_NE );
}

```

```

[\.]{n}{e}{q}{v}{\.} {
#ifdef DEBUG

```

```

    ECHO;
#endif
    return( RW_NEQV );
}

[\.]{n}{o}{t}{\.} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_NOT );
}

[\.]{o}{r}{\.} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_OR );
}

[\.]{t}{r}{u}{e}{\.} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_TRUE );
}

{a}{s}{s}{i}{g}{n} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_ASSIGN );
}

{b}{a}{c}{k}{s}{p}{a}{c}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_BACKSPACE );
}

{b}{l}{o}{c}{k}{\ }*{d}{a}{t}{a} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_BLOCK_DATA );
}

{c}{a}{l}{l} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_CALL );
}

{c}{h}{a}{r}{a}{c}{t}{e}{r} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_CHARACTER );
}

{c}{l}{o}{s}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_CLOSE );
}

{c}{o}{m}{m}{o}{n} {
#ifdef DEBUG
    ECHO;

```



```

#endif
    return( RW_COMMON );
}

{c}{o}{m}{p}{l}{e}{x} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_COMPLEX );
}

{c}{o}{n}{t}{i}{n}{u}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_CONTINUE );
}

{d}{a}{t}{a} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_DATA );
}

{d}{i}{m}{e}{n}{s}{i}{o}{n} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_DIMENSION );
}

{d}{o} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_DO );
}

{d}{o}{u}{b}{l}{e}{\ }*{p}{r}{e}{c}{i}{s}{i}{o}{n} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_DOUBLE_PRECISION );
}

{e}{l}{s}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_ELSE );
}

{e}{l}{s}{e}{\ }*{i}{f} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_ELSE_IF );
}

{e}{n}{d} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_END );
}

{e}{n}{d}{\ }*{i}{f} {
#ifdef DEBUG
    ECHO;
#endif
}

```

```

        return( RW_END_IF );
    }

{e}{n}{d}{f}{i}{l}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_ENDFILE );
}

{e}{n}{t}{r}{y} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_ENTRY );
}

{e}{q}{u}{i}{v}{a}{l}{e}{n}{c}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_EQUIVALENCE );
}

{e}{x}{t}{e}{r}{n}{a}{l} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_EXTERNAL );
}

{f}{o}{r}{m}{a}{t}.* {
#ifdef DEBUG
    ECHO;
#endif
    yylval = duplicate( yytext );
    return( RW_FORMAT );
}

{f}{u}{n}{c}{t}{i}{o}{n} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_FUNCTION );
}

{g}{o}{\ }*{t}{o} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_GO_TO );
}

{i}{f} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_IF );
}

{i}{m}{p}{l}{i}{c}{i}{t} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_IMPLICIT );
}

{i}{n}{c}{l}{u}{d}{e} {
#ifdef DEBUG
    ECHO;
#endif

```

```

        return( RW_INCLUDE );
    }

    {i}{n}{q}{u}{i}{r}{e} {
#ifdef DEBUG
        ECHO;
#endif
        return( RW_INQUIRE );
    }

    {i}{n}{t}{e}{g}{e}{r} {
#ifdef DEBUG
        ECHO;
#endif
        return( RW_INTEGER );
    }

    {i}{n}{t}{r}{i}{n}{s}{i}{c} {
#ifdef DEBUG
        ECHO;
#endif
        return( RW_INTRINSIC );
    }

    {l}{o}{g}{i}{c}{a}{l} {
#ifdef DEBUG
        ECHO;
#endif
        return( RW_LOGICAL );
    }

    {n}{a}{m}{e}{l}{i}{s}{t} {
#ifdef DEBUG
        ECHO;
#endif
        return( RW_NAMELIST );
    }

    {o}{p}{e}{n} {
#ifdef DEBUG
        ECHO;
#endif
        return( RW_OPEN );
    }

    {p}{a}{r}{a}{m}{e}{t}{e}{r} {
#ifdef DEBUG
        ECHO;
#endif
        return( RW_PARAMETER );
    }

    {p}{a}{u}{s}{e} {
#ifdef DEBUG
        ECHO;
#endif
        return( RW_PAUSE );
    }

    {p}{r}{i}{n}{t} {
#ifdef DEBUG
        ECHO;
#endif
        return( RW_PRINT );
    }

    {p}{r}{o}{g}{r}{a}{m} {
#ifdef DEBUG
        ECHO;
#endif
        return( RW_PROGRAM );
    }

```

```

    }

{r}{e}{a}{d} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_READ );
}

{r}{e}{a}{l} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_REAL );
}

{r}{e}{t}{u}{r}{n} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_RETURN );
}

{r}{e}{w}{i}{n}{d} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_REWIND );
}

{s}{a}{v}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_SAVE );
}

{s}{t}{o}{p} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_STOP );
}

{s}{u}{b}{r}{o}{u}{t}{i}{n}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_SUBROUTINE );
}

{t}{h}{e}{n} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_THEN );
}

{t}{o} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_TO );
}

{w}{r}{i}{t}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_WRITE );
}

```

```

{u}{n}{d}{e}{f}{i}{n}{e}{d} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_UNDEFINED );
}

[a-zA-Z][_a-zA-Z0-9]* {
#ifdef DEBUG
    ECHO;
#endif
    yylval = duplicate( uppercase( yytext ) );
    return( IDENTIFIER );
}

^[0-9][0-9][0-9][0-9][0-9][\ ] {
#ifdef DEBUG
    ECHO;
#endif
    yylval = duplicate( non_blank( yytext ) );
    return( LABEL );
}

[0-9]+ |
[0-9]+/\.[a-zA-Z]+\.[\ ] {
#ifdef DEBUG
    ECHO;
#endif
    yylval = duplicate( yytext );
    return( INTEGER );
}

[0-9]+\.[0-9]*([eE][\+\-]?[0-9]+)? |
[0-9]*\.[0-9]+([eE][\+\-]?[0-9]+)? |
[0-9]+([eE][\+\-]?[0-9]+){
#ifdef DEBUG
    ECHO;
#endif
    yylval = duplicate( yytext );
    return( REAL );
}

[0-9]+\.[0-9]*([dD][\+\-]?[0-9]+)? |
[0-9]*\.[0-9]+([dD][\+\-]?[0-9]+)? |
[0-9]+([dD][\+\-]?[0-9]+){
#ifdef DEBUG
    ECHO;
#endif
    yylval = duplicate( yytext );
    return( DOUBLE_PRECISION );
}

\[^\']*\\' |
\[^\"]*\\\" {
#ifdef DEBUG
    ECHO;
#endif
    yytext[ 0 ] = '\\';
    yytext[ strlen( yytext ) - 1 ] = '\\';
    yylval = duplicate( yytext );
    return( STRING );
}

[0-9]+[hH] {
#ifdef DEBUG
    ECHO;
#endif
    yylval = duplicate( hollerith( yytext, '\\\"' ) );
    return( HOLLERITH );
}

```

FILE: equivalence/statement/Makefile

```
#
# Copyright 1991
# Georgia Institute of Technology
# Computer Engineering Research Laboratory
# Author: Stephen R. Wachtel
#
```

```
CC = cc -g
INCLUDE = ../include
CFLAGS = -IS(INCLUDE)
LIBRARY = statement.a
```

```
OBJECTS = \
    data_statement.o
```

```
$(LIBRARY): $(OBJECTS)
    rm -f $(LIBRARY)
    ar crv $(LIBRARY) $(OBJECTS)
    ranlib $(LIBRARY)
```

```
.SUFFIXES: .c .o
.c.o:
    $(CC) -c $(CFLAGS) $<
```

```
clean:
    rm -f $(LIBRARY) $(OBJECTS)
```

FILE: equivalence/statement/data\_statement.c

```
/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */
```

```
#include <stdio.h>
#include "list.h"
```

```
extern LIST *rlist;
extern LIST *ilist;
extern char *merge( );
extern char *parse( );
```

```
void data_statement( data_list )
register Char *data_list;
{
    register char *data;
    register char *variable_list;
    register char *constant_list;

    register char *variable;
    register char *constant;

    register LIST *temporary;
    register int index;

    while ( data = parse( data_list ) )
    {
        variable_list = parse( data );
        constant_list = parse( data );

        variable = parse( variable_list );

        if ( strcmp( variable, "REAL", 5 ) == 0 )
        {
            index = 0;
            while ( constant = parse( constant_list ) )
```

```

        {
            if ( atoi( constant ) != 0 )
            {
                if ( ( temporary = find_index( rlist, ++index ) ) != 0 )
                    temporary->alternate = merge( "RIN(%s)", constant );
                else
                    printf( "ERROR: rlist(%d) not found\n", index );
            }
        }
    }

    if ( strcmp( variable, "IINT", 4 ) == 0 )
    {
        index = 0;
        while ( constant = parse( constant_list ) )
        {
            if ( atoi( constant ) != 0 )
            {
                if ( ( temporary = find_index( ilist, ++index ) ) != 0 )
                    temporary->alternate = merge( "IIN(%s)", constant );
                else
                    printf( "ERROR: ilist(%d) not found\n", index );
            }
        }
    }
}
/* data_statement */

```

### 13. Appendix H: etimer program source

FILE: etimer/Makefile

```
#
# Copyright 1991
# Georgia Institute of Technology
# Computer Engineering Research Laboratory
# Author: Stephen R. Wachtel
#

default:    etimer

CC = cc -g
INCLUDE = include
CFLAGS = -I$(INCLUDE)
LIBRARY = statement/statement.a library/library.a

OBJECTS = \
$(INCLUDE)/grammar.h \
*grammar.[co] \
*scanner.[co] \
yytrace.[co] \
y.output

PROGRAMS = \
*etimer

grammar.c: grammar.y
yacc -dv grammar.y
mv y.tab.h $(INCLUDE)/grammar.h
mv y.tab.c grammar.c

scanner.c: scanner.l
lex -vt scanner.l | sed 's/getc/yygetc/' >scanner.c

scanner.o: scanner.c $(INCLUDE)/grammar.h
$(CC) $(CFLAGS) -c scanner.c

grammar.o: grammar.c
$(CC) $(CFLAGS) -c grammar.c

etimer: grammar.o scanner.o $(LIBRARY)
$(CC) -o etimer grammar.o scanner.o $(LIBRARY)

sgrammar.c: grammar.c yytoken.awk
awk -f yytoken.awk <grammar.c >sgrammar.c

sgrammar.o: sgrammar.c
$(CC) $(CFLAGS) -c sgrammar.c

setimer:    sgrammar.o scanner.o $(LIBRARY)
$(CC) -o setimer sgrammar.o scanner.o $(LIBRARY)

dscanner.c: scanner.c
cp scanner.c dscanner.c

dscanner.o: dscanner.c $(INCLUDE)/grammar.h
$(CC) $(CFLAGS) -DDEBUG -c dscanner.c

detimer:    grammar.o dscanner.o $(LIBRARY)
$(CC) -o detimer grammar.o dscanner.o $(LIBRARY)

tgrammar.c: grammar.c
sed 's/yystack:/& yytrace(yystate);/' <grammar.c >tgrammar.c

tgrammar.o: tgrammar.c
$(CC) $(CFLAGS) -c tgrammar.c
```



```
tetimer:  tgrammar.o scanner.o yytrace.o $(LIBRARY)
          $(CC) -o tetimer tgrammar.o scanner.o yytrace.o $(LIBRARY)
```

```
yytrace.c: grammar.c yytrace.awk
          awk -f yytrace.awk <y.output >yytrace.c
```

```
yytrace.o: yytrace.c
          $(CC) $(CFLAGS) -c yytrace.c
```

```
clean:
        cd statement; make clean
        cd library; make clean
        rm -f $(PROGRAMS) $(OBJECTS)
```

```
FILE: etimer/grammar.y
```

```
/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author:  Stephen R. Wachtel
 */
```

```
/*
 * FORTRAN 77
 */
```

```
%token RW_AND
%token RW_ASSIGN
%token RW_BACKSPACE
%token RW_BLOCK_DATA
%token RW_CALL
%token RW_CHARACTER
%token RW_CLOSE
%token RW_COMMON
%token RW_COMPLEX
%token RW_CONTINUE
%token RW_DATA
%token RW_DIMENSION
%token RW_DO
%token RW_DOUBLE_PRECISION
%token RW_ELSE
%token RW_ELSE_IF
%token RW_END
%token RW_END_IF
%token RW_ENDFILE
%token RW_ENTRY
%token RW_EQ
%token RW_EQUIVALENCE
%token RW_EQV
%token RW_EXTERNAL
%token RW_FALSE
%token RW_FORMAT
%token RW_FUNCTION
%token RW_GE
%token RW_GO_TO
%token RW_GT
%token RW_IF
%token RW_IMPLICIT
%token RW_INCLUDE
%token RW_INQUIRE
%token RW_INTEGER
%token RW_INTRINSIC
%token RW_LE
%token RW_LOGICAL
%token RW_LT
%token RW_NAMELIST
%token RW_NE
%token RW_NEQV
%token RW_NOT
%token RW_OPEN
%token RW_OR
%token RW_PARAMETER
%token RW_PAUSE
%token RW_PRINT
```

```

%token RW_PROGRAM
%token RW_READ
%token RW_REAL
%token RW_RETURN
%token RW_REWIND
%token RW_SAVE
%token RW_TOP
%token RW_SUBROUTINE
%token RW_THEN
%token RW_TO
%token RW_TRUE
%token RW_WRITE
%token RW_UNDEFED

%token COMMENT
%token CONCATENATE
%token DOUBLE_PRECISION
%token EXPONENTIATE
%token HOLLERITH
%token IDENTIFIER
%token INTEGER
%token LABEL
%token REAL
%token STRING

%left ','
%nonassoc ':'
%right '='
%left RW_EQV RW_NEQV
%left RW_OR
%left RW_AND
%left RW_NOT
%nonassoc RW_EQ RW_NE RW_LT RW_LE RW_GT RW_GE
%left CONCATENATE
%left '+' '-'
%left '*' '/'
%right EXPONENTIATE
%left SIGN

%{
typedef char *POINTER;
#define YYSTYPE POINTER

extern POINTER array( );
extern POINTER duplicate( );
extern POINTER implied_do_list( );
extern POINTER label( );
extern POINTER list( );
extern POINTER merge( );
extern POINTER type( );
%}

%%

program:
    optional_statement_list
    {
        summary( );
    }
    ;

optional_statement_list:
    /* NULL */
    |
    statement_list
    ;

statement_list:
    statement
    |
    statement_list statement
    ;

```

```

statement:
    comment_statement
    |
    label_unlabeled_statement
    {
        statement( $1 );
    }
    ;

comment_statement:
    COMMENT
    {
        if ( timer( $1 ) == 0 )
        {
            comment_statement( $1 );
        }
    }
    ;

label:
    LABEL
    {
        $$ = label( $1 );
    }
    ;

unlabeled_statement:
    include_statement
    |
    program_statement
    |
    block_data_statement
    |
    function_statement
    |
    subroutine_statement
    |
    entry_statement
    |
    end_statement
    |
    specification_statement
    |
    executable_statement
    |
    format_statement
    ;

include_statement:
    RW_INCLUDE character_constant
    {
        include_statement( $2 );
    }
    ;

program_statement:
    RW_PROGRAM program_identifier
    {
        program( $2 );
        program_statement( $2 );
    }
    ;

program_identifier:
    IDENTIFIER
    {
        $$ = $1;
    }
    ;

block_data_statement:
    RW_BLOCK_DATA block_data_identifier
    {
        block_data_statement( $2 );
    }

```

```

    }
;

block_data_identifier:
    IDENTIFIER
    {
        $$ = $1;
    }
;

function_statement:
    RW_FUNCTION function_identifier optional_formal_argument_list
    {
        function_statement( 0, $2, $3 );
    }
|
    type RW_FUNCTION function_identifier optional_formal_argument_list
    {
        function_statement( $1, $3, $4 );
    }
;

function_identifier:
    IDENTIFIER
    {
        $$ = $1;
    }
;

subroutine_statement:
    RW_SUBROUTINE subroutine_identifier
    {
        subroutine_statement( $2, 0 );
    }
|
    RW_SUBROUTINE subroutine_identifier optional_formal_argument_list
    {
        subroutine_statement( $2, $3 );
    }
;

subroutine_identifier:
    IDENTIFIER
    {
        $$ = $1;
    }
;

entry_statement:
    RW_ENTRY entry_identifier
    {
        entry_statement( $2, 0 );
    }
|
    RW_ENTRY entry_identifier optional_formal_argument_list
    {
        entry_statement( $2, $3 );
    }
;

entry_identifier:
    IDENTIFIER
    {
        $$ = $1;
    }
;

optional_formal_argument_list:
    '(', ' ', ')'
    {
        $$ = 0;
    }
|

```

```

    (' formal_argument_list ')
    {
        $$ = $2;
    }
;

formal_argument_list:
    formal_argument
    {
        $$ = merge( "${s}", $1 );
    }
    |
    formal_argument_list ' ' formal_argument
    {
        $$ = merge( "${s}${s}", $1, $3 );
    }
;

formal_argument:
    IDENTIFIER
    {
        $$ = $1;
    }
    |
    formal_argument_alternate_return
    {
        $$ = $1;
    }
;

formal_argument_alternate_return:
    '*'
    {
        $$ = duplicate( "*" );
    }
;

end_statement:
    RW_END
    {
        end_statement( );
    }
;

specification_statement:
    external_statement
    |
    intrinsic_statement
    |
    parameter_statement
    |
    dimension_statement
    |
    declaration_statement
    |
    save_statement
    |
    common_statement
    |
    equivalence_statement
    |
    implicit_statement
    |
    data_statement
    |
    namelist_statement
;

external_statement:
    RW_EXTERNAL external_list
    {
        external_statement( $2 );
    }
;

```

```

external_list:
    external
    {
        $$ = merge( "{%s}", $1 );
    }
    |
    external_list ',' external
    {
        $$ = merge( "%s{%s}", $1, $3 );
    }
    ;

external:
    IDENTIFIER
    {
        $$ = $1;
    }
    ;

intrinsic_statement:
    RW_INTRINSIC intrinsic_list
    {
        intrinsic_statement( $2 );
    }
    ;

intrinsic_list:
    intrinsic
    {
        $$ = merge( "{%s}", $1 );
    }
    |
    intrinsic_list ',' intrinsic
    {
        $$ = merge( "%s{%s}", $1, $3 );
    }
    ;

intrinsic:
    IDENTIFIER
    {
        $$ = $1;
    }
    ;

parameter_statement:
    RW_PARAMETER '(' parameter_list ')'
    {
        parameter_statement( $3 );
    }
    ;

parameter_list:
    parameter
    {
        $$ = merge( "{%s}", $1 );
    }
    |
    parameter_list ',' parameter
    {
        $$ = merge( "%s{%s}", $1, $3 );
    }
    ;

parameter:
    IDENTIFIER '=' expression
    {
        $$ = merge( "{%s}{%s}", $1, $3 );
    }
    ;

dimension_statement:

```

```

RW_DIMENSION dimension_list
{
    dimension_statement( $2 );
}
;

dimension_list:
dimension
{
    $$ = merge( "{%s}", $1 );
}
|
dimension_list ',' dimension
{
    $$ = merge( "%s{%s}", $1, $3 );
}
;

dimension:
IDENTIFIER '(' subscript_list ')'
{
    $$ = merge( "{%s}{%s}", $1, $3 );
}
;

subscript_list:
subscript
{
    $$ = merge( "{%s}", $1 );
}
|
subscript_list ',' subscript
{
    $$ = merge( "%s{%s}", $1, $3 );
}
;

subscript:
upper_bound
{
    $$ = $1;
}
|
lower_bound ':' upper_bound
{
    $$ = merge( "%s:%s", $1, $3 );
}
;

lower_bound:
expression
{
    $$ = $1;
}
;

upper_bound:
lower_bound
{
    $$ = $1;
}
|
upper_bound_adjustable
{
    $$ = $1;
}
;

upper_bound_adjustable:
'..'
{
    $$ = duplicate( "" );
}
;

```

```

declaration_statement:
    type_declaration_list
    {
        declaration_statement( $1, $2 );
    }
;

declaration_list:
    declaration
    {
        $$ = merge( "{%s}", $1 );
    }
|
    declaration_list ',' declaration
    {
        $$ = merge( "%s{%s}", $1, $3 );
    }
;

declaration:
    IDENTIFIER
    {
        $$ = merge( "{%s}", $1 );
    }
|
    IDENTIFIER '(' subscript_list ')'
    {
        $$ = merge( "{%s}{%s}", $1, $3 );
    }
;

type:
    type_name optional_type_length
    {
        $$ = type( $1, $2 );
    }
;

type_name:
    RW_CHARACTER
    {
        $$ = duplicate( "CHARACTER" );
    }
|
    RW_COMPLEX
    {
        $$ = duplicate( "COMPLEX" );
    }
|
    RW_DOUBLE_PRECISION
    {
        $$ = duplicate( "DOUBLE PRECISION" );
    }
|
    RW_INTEGER
    {
        $$ = duplicate( "INTEGER" );
    }
|
    RW_LOGICAL
    {
        $$ = duplicate( "LOGICAL" );
    }
|
    RW_REAL
    {
        $$ = duplicate( "REAL" );
    }
|
    RW_UNDEFINED
    {
        $$ = duplicate( "UNDEFINED" );
    }
;

```



```

optional_type_length:
/* NULL */
{
    $$ = 0;
}
|
type_length
{
    $$ = $1;
}
;

type_length:
/* INTEGER
{
    $$ = merge( "%s", $2 );
}
|
/* type_length_adjustable
{
    $$ = merge( "%s", $2 );
}
;

type_length_adjustable:
/* ('*' ')
{
    $$ = duplicate( "(" );
}
;

save_statement:
RW_SAVE optional_save_list
{
    save_statement( $2 );
}
;

optional_save_list:
/* NULL */
{
    $$ = 0;
}
|
save_list
{
    $$ = $1;
}
;

save_list:
save
{
    $$ = merge( "%s", $1 );
}
|
save_list ',' save
{
    $$ = merge( "%s%s", $1, $3 );
}
;

save:
IDENTIFIER
{
    $$ = $1;
}
|
common_name
{
    $$ = $1;
}
;

```

```

common_statement:
    RW_COMMON optional_common_name common_list
    {
        common_statement( $2, $3 );
    }
;

```

```

optional_common_name:
    /* NULL */
    {
        $$ = 0;
    }
    |
    common_name
    {
        $$ = $1;
    }
;

```

```

common_name:
    '/' optional_identifier '/'
    {
        $$ = $2;
    }
;

```

```

optional_identifier:
    /* NULL */
    {
        $$ = 0;
    }
    |
    IDENTIFIER
    {
        $$ = $1;
    }
;

```

```

common_list:
    common
    {
        $$ = merge( "%s", $1 );
    }
    |
    common_list ',' common
    {
        $$ = merge( "%s%s", $1, $3 );
    }
;

```

```

common:
    IDENTIFIER
    {
        $$ = merge( "%s", $1 );
    }
    |
    IDENTIFIER '(' subscript_list ')'
    {
        $$ = merge( "%s{%s}", $1, $3 );
    }
;

```

```

equivalence_statement:
    RW_EQUIVALENCE equivalence_list
    {
        equivalence_statement( $2 );
    }
;

```

```

equivalence_list:
    equivalence
    {
        $$ = merge( "%s", $1 );
    }
;

```

```

    }
    |
    equivalence_list ',' equivalence
    {
        $$ = merge( "%s{%s}", $1, $3 );
    }
    ;

equivalence:
    '(' variable_list ')'
    {
        $$ = $2;
    }
    ;

variable_list:
    variable
    {
        $$ = merge( "{%s}", $1 );
    }
    |
    variable_list ',' variable
    {
        $$ = merge( "%s{%s}", $1, $3 );
    }
    ;

implicit_statement:
    RW_IMPLICIT type '(' implicit_list ')'
    {
        implicit_statement( $2, $4 );
    }
    ;

implicit_list:
    implicit
    {
        $$ = merge( "{%s}", $1 );
    }
    |
    implicit_list ',' implicit
    {
        $$ = merge( "%s{%s}", $1, $3 );
    }
    ;

implicit:
    IDENTIFIER
    {
        $$ = $1;
    }
    |
    IDENTIFIER '-' IDENTIFIER
    {
        $$ = merge( "%s-%s", $1, $3 );
    }
    ;

namelist_statement:
    RW_NAMELIST namelist_name namelist_list
    {
        namelist_statement( $2, $3 );
    }
    ;

namelist_name:
    '/' IDENTIFIER '/'
    {
        $$ = $2;
    }
    ;

namelist_list:

```

```

    namelist
    {
        $$ = merge( "{%s}", $1 );
    }
|
    namelist_list ',' namelist
    {
        $$ = merge( "%s{%s}", $1, $3 );
    }
;

namelist:
    IDENTIFIER
    {
        $$ = $1;
    }
;

data_statement:
    RW_DATA data_list
    {
        data_statement( $2 );
    }
;

data_list:
    data
    {
        $$ = merge( "{%s}", $1 );
    }
|
    data_list optional_comma data
    {
        $$ = merge( "%s{%s}", $1, $3 );
    }
;

data:
    data_variable_list '/' data_constant_list '/'
    {
        $$ = merge( "{%s}{%s}", $1, $3 );
    }
;

data_variable_list:
    data_variable
    {
        $$ = merge( "{%s}", $1 );
    }
|
    data_variable_list ',' data_variable
    {
        $$ = merge( "%s{%s}", $1, $3 );
    }
;

data_variable:
    variable
    {
        $$ = $1;
    }
|
    data_implied_do_list
    {
        $$ = $1;
    }
;

data_implied_do_list:
    '(' data_variable_list ',' IDENTIFIER '=' expression_list ')'
    {
        $$ = implied_do_list( $2, $4, $6 );
    }
;

```

```

data_constant_list:
  data_constant
  {
    $$ = merge( "%s", $1 );
  }
|
  data_constant_list ',' data_constant
  {
    $$ = merge( "%s%s", $1, $3 );
  }
;

```

```

data_constant:
  data_initialization
  {
    $$ = $1;
  }
|
  IDENTIFIER '*' data_initialization
  {
    $$ = merge( "%s * %s", $1, $3 );
  }
|
  INTEGER '*' data_initialization
  {
    $$ = merge( "%s * %s", $1, $3 );
  }
;

```

```

data_initialization:
  IDENTIFIER
  {
    $$ = $1;
  }
|
  character_constant
  {
    $$ = $1;
  }
|
  logical_constant
  {
    $$ = $1;
  }
|
  signed_numerical_constant
  {
    $$ = $1;
  }
;

```

```

signed_numerical_constant:
  numerical_constant
  {
    $$ = $1;
  }
|
  '+' numerical_constant %prec SIGN
  {
    $$ = merge( "+%s", $2 );
  }
|
  '-' numerical_constant %prec SIGN
  {
    $$ = merge( "-%s", $2 );
  }
;

```

```

expression:
  parenthesis_expression
  {
    $$ = $1;
  }
|
  simple_expression

```

```

    {
        $$ = $1;
    }
;

parenthesis_expression:
    '(' expression ')'
    {
        $$ = merge( "(", $2 );
    }
;

simple_expression:
    variable
    {
        $$ = $1;
    }
    |
    constant
    {
        $$ = $1;
    }
    |
    arithmetic_expression
    {
        $$ = $1;
    }
    |
    character_expression
    {
        $$ = $1;
    }
    |
    relational_expression
    {
        $$ = $1;
    }
    |
    logical_expression
    {
        $$ = $1;
    }
    |
    unary_expression
    {
        $$ = $1;
    }
;

variable:
    IDENTIFIER
    {
        $$ = $1;
    }
    |
    IDENTIFIER string_subset
    {
        $$ = merge( "%s %s", $1, $2 );
    }
    |
    array
    {
        $$ = $1;
    }
;

array:
    IDENTIFIER '(' optional_expression_list ')'
    {
        $$ = array( $1, $3 );
    }
    |
    IDENTIFIER '(' optional_expression_list ')' string_subset
    {
        $$ = merge( "%s %s", array( $1, $3 ), $5 );
    }
;

```

```
optional_expression_list:
```

```
    /* NULL */
    {
        $$ = 0;
    }
    |
    expression_list
    {
        $$ = $1;
    }
    ;
```

```
expression_list:
```

```
    expression
    {
        $$ = merge( "%s)", $1 );
    }
    |
    expression_list ',' expression
    {
        $$ = merge( "%s{%s}", $1, $3 );
    }
    ;
```

```
string_subset:
```

```
    '(' optional_expression ':' optional_expression ')'
    {
        $$ = merge( "%s:%s)", $2, $4 );
    }
    ;
```

```
optional_expression:
```

```
    /* NULL */
    {
        $$ = 0;
    }
    |
    expression
    {
        $$ = $1;
    }
    ;
```

```
constant:
```

```
    character_constant
    {
        $$ = $1;
    }
    |
    logical_constant
    {
        $$ = $1;
    }
    |
    numerical_constant
    {
        $$ = $1;
    }
    ;
```

```
character_constant:
```

```
    HOLLERITH
    {
        $$ = $1;
    }
    |
    STRING
    {
        $$ = $1;
    }
    ;
```

```
logical_constant:
```

```

    RW_FALSE
    {
        $$ = duplicate( ".FALSE." );
    }
|
    RW_TRUE
    {
        $$ = duplicate( ".TRUE." );
    }
;

numerical_constant:
    DOUBLE_PRECISION
    {
        $$ = $1;
    }
|
    INTEGER
    {
        $$ = $1;
    }
|
    REAL
    {
        $$ = $1;
    }
;

arithmetic_expression:
    expression '+' expression %prec '+'
    {
        $$ = merge( "%s + %s", $1, $3 );
    }
|
    expression '-' expression %prec '-'
    {
        $$ = merge( "%s - %s", $1, $3 );
    }
|
    expression '*' expression %prec '*'
    {
        $$ = merge( "%s * %s", $1, $3 );
    }
|
    expression '/' expression %prec '/'
    {
        $$ = merge( "%s / %s", $1, $3 );
    }
|
    expression EXPONENTIATE expression %prec EXPONENTIATE
    {
        $$ = merge( "%s ** %s", $1, $3 );
    }
;

character_expression:
    expression '/' '/' expression %prec CONCATENATE
    {
        $$ = merge( "%s // %s", $1, $4 );
    }
;

relational_expression:
    expression RW_EQ expression %prec RW_EQ
    {
        $$ = merge( "%s .EQ. %s", $1, $3 );
    }
|
    expression RW_NE expression %prec RW_NE
    {
        $$ = merge( "%s .NE. %s", $1, $3 );
    }
|
    expression RW_LT expression %prec RW_LT
    {
        $$ = merge( "%s .LT. %s", $1, $3 );
    }

```



```

|
| expression RW_LE expression %prec RW_LE
| {
|     $$ = merge( "%s .LE. %s", $1, $3 );
| }
|
| expression RW_GT expression %prec RW_GT
| {
|     $$ = merge( "%s .GT. %s", $1, $3 );
| }
|
| expression RW_GE expression %prec RW_GE
| {
|     $$ = merge( "%s .GE. %s", $1, $3 );
| }
;

logical_expression:
| expression RW_AND expression %prec RW_AND
| {
|     $$ = merge( "%s .AND. %s", $1, $3 );
| }
|
| expression RW_OR expression %prec RW_OR
| {
|     $$ = merge( "%s .OR. %s", $1, $3 );
| }
|
| expression RW_EQV expression %prec RW_EQV
| {
|     $$ = merge( "%s .EQV. %s", $1, $3 );
| }
|
| expression RW_NEQV expression %prec RW_NEQV
| {
|     $$ = merge( "%s .NEQV. %s", $1, $3 );
| }
;

unary_expression:
| '+' expression %prec SIGN
| {
|     $$ = merge( "+%s", $2 );
| }
|
| '-' expression %prec SIGN
| {
|     $$ = merge( "-%s", $2 );
| }
|
| RW_NOT expression %prec RW_NOT
| {
|     $$ = merge( ".NOT. %s", $2 );
| }
;

executable_statement:
| do_statement
|
| logical_if_statement
|
| block_if_statement
|
| else_statement
|
| else_if_statement
|
| end_if_statement
|
| subset_executable_statement
;

do_statement:
| RW_DO INTEGER IDENTIFIER '=' expression_list
| {
|     computation( 0 );
|     do_statement( $2, $3, $5 );
| }

```

```

    }
;

logical_if_statement:
    if_expression subset_executable_statement
    {
        logical_if_statement( );
    }
;

if_expression:
    RW_IF '(' expression ')'
    {
        computation( 0 );
        if_expression( $3 );
    }
;

block_if_statement:
    RW_IF '(' expression ')' RW_THEN
    {
        computation( 0 );
        block_if_statement( $3 );
    }
;

else_statement:
    RW_ELSE
    {
        computation( 0 );
        else_statement( );
    }
;

else_if_statement:
    RW_ELSE_IF '(' expression ')' RW_THEN
    {
        computation( 0 );
        else_if_statement( $3 );
    }
;

end_if_statement:
    RW_END_IF
    {
        computation( 0 );
        end_if_statement( );
    }
;

subset_executable_statement:
    assignment_statement
    |
    assign_statement
    |
    arithmetic_if_statement
    |
    continue_statement
    |
    call_statement
    |
    return_statement
    |
    unconditional_go_to_statement
    |
    computed_go_to_statement
    |
    assigned_go_to_statement
    |
    stop_statement
    |
    pause_statement
    |
    io_statement

```

```

;

assignment_statement:
    variable '=' expression
    {
        computation( 0 );
        assignment_statement( $1, $3 );
    }
;

assign_statement:
    RW_ASSIGN INTEGER RW_TO IDENTIFIER
    {
        computation( 0 );
        assign_statement( $2, $4 );
    }
;

arithmetic_if_statement:
    RW_IF '(' expression ')' integer_list
    {
        computation( 0 );
        arithmetic_if_statement( $3, $5 );
    }
;

continue_statement:
    RW_CONTINUE
    {
        computation( 0 );
        continue_statement( );
    }
;

call_statement:
    RW_CALL IDENTIFIER
    {
        computation( $2 );
        call_statement( $2, 0 );
    }
    |
    RW_CALL IDENTIFIER optional_actual_argument_list
    {
        if ( ( strcmp( $2, "send ", strlen( "send " ) ) == 0 )
            || ( strcmp( $2, "receive ", strlen( "receive " ) ) == 0 ) )
            communication( $2, list( duplicate( $3 ), ", " ) );
        else
            computation( $2 );
        call_statement( $2, $3 );
    }
;

optional_actual_argument_list:
    '(' ')'
    {
        $$ = 0;
    }
    |
    '(' actual_argument_list ')'
    {
        $$ = $2;
    }
;

actual_argument_list:
    actual_argument
    {
        $$ = merge( "%s", $1 );
    }
    |
    actual_argument_list ',' actual_argument
    {
        $$ = merge( "%s%s", $1, $3 );
    }

```

```

    }
;

actual_argument:
    expression
    {
        $$ = $1;
    }
|
    actual_argument_alternate_return
    {
        $$ = $1;
    }
;

actual_argument_alternate_return:
    '*' INTEGER
    {
        $$ = merge( "**%s", $2 );
    }
;

return_statement:
    RW_RETURN optional_expression
    {
        computation( 0 );
        return_statement( $2 );
    }
;

unconditional_go_to_statement:
    RW_GO_TO INTEGER
    {
        computation( 0 );
        unconditional_go_to_statement( $2 );
    }
;

computed_go_to_statement:
    RW_GO_TO '(' integer_list ')' optional_comma expression
    {
        computation( 0 );
        computed_go_to_statement( $3, $6 );
    }
;

assigned_go_to_statement:
    RW_GO_TO IDENTIFIER
    {
        computation( 0 );
        assigned_go_to_statement( $2, 0 );
    }
|
    RW_GO_TO IDENTIFIER optional_comma '(' integer_list ')'
    {
        computation( 0 );
        assigned_go_to_statement( $2, $5 );
    }
;

optional_comma:
    /* NULL */
|
    ','
;

integer_list:
    INTEGER
    {
        $$ = merge( "(%s)", $1 );
    }
|
    integer_list ',' INTEGER

```

```

    {
        $$ = merge( "%s{%s}", $1, $3 );
    }
;

pause_statement:
    RW_PAUSE optional_expression
    {
        computation( 0 );
        pause_statement( $2 );
    }
;

stop_statement:
    RW_STOP optional_expression
    {
        computation( 0 );
        stop_statement( $2 );
    }
;

io_statement:
    open_statement
    |
    close_statement
    |
    inquire_statement
    |
    read_statement
    |
    write_statement
    |
    print_statement
    |
    backspace_statement
    |
    rewind_statement
    |
    endfile_statement
;

open_statement:
    RW_OPEN '(' control_information_list ')'
    {
        computation( 0 );
        open_statement( $3 );
    }
;

close_statement:
    RW_CLOSE '(' control_information_list ')'
    {
        computation( 0 );
        close_statement( $3 );
    }
;

inquire_statement:
    RW_INQUIRE '(' control_information_list ')'
    {
        computation( 0 );
        inquire_statement( $3 );
    }
;

read_statement:
    RW_READ '(' control_information_list ')' optional_io_list
    {
        computation( 0 );
        read_statement( $3, $5 );
    }
    |
    RW_READ control
    {

```

```

        computation( 0 );
        read_statement( $2, 0 );
    }
|
    RW_READ control ',' io_list
    {
        computation( 0 );
        read_statement( $2, $4 );
    }
;

write_statement:
    RW_WRITE '(' control_information_list ')' optional_io_list
    {
        computation( 0 );
        write_statement( $3, $5 );
    }
;

print_statement:
    RW_PRINT control
    {
        computation( 0 );
        print_statement( $2, 0 );
    }
|
    RW_PRINT control ',' io_list
    {
        computation( 0 );
        print_statement( $2, $4 );
    }
;

backspace_statement:
    RW_BACKSPACE '(' control_information_list ')'
    {
        computation( 0 );
        backspace_statement( $3 );
    }
|
    RW_BACKSPACE control
    {
        computation( 0 );
        backspace_statement( $2 );
    }
;

rewind_statement:
    RW_REWIND '(' control_information_list ')'
    {
        computation( 0 );
        rewind_statement( $3 );
    }
|
    RW_REWIND control
    {
        computation( 0 );
        rewind_statement( $2 );
    }
;

endfile_statement:
    RW_ENDFILE '(' control_information_list ')'
    {
        computation( 0 );
        endfile_statement( $3 );
    }
|
    RW_ENDFILE control
    {
        computation( 0 );
        endfile_statement( $2 );
    }
;

```

```

control_information_list:
  control_information
  {
    $$ = merge( "{%s}", $1 );
  }
  |
  control_information_list ',' control_information
  {
    $$ = merge( "%s{%s}", $1, $3 );
  }
;

```

```

control_information:
  control
  {
    $$ = $1;
  }
  |
  IDENTIFIER '=' expression
  {
    $$ = merge( "%s = %s", $1, $3 );
  }
;

```

```

control:
  variable
  {
    $$ = $1;
  }
  |
  constant
  {
    $$ = $1;
  }
  |
  '*'
  {
    $$ = duplicate( "*" );
  }
;

```

```

optional_io_list:
  /* NULL */
  {
    $$ = 0;
  }
  |
  io_list
  {
    $$ = $1;
  }
;

```

```

io_list:
  io
  {
    $$ = merge( "{%s}", $1 );
  }
  |
  io_list ',' io
  {
    $$ = merge( "%s{%s}", $1, $3 );
  }
;

```

```

io:
  expression
  {
    $$ = $1;
  }
  |
  io_implied_do_list
  {
    $$ = $1;
  }
;

```

```

io_implied_do_list:
    '(' io_list ',' IDENTIFIER '=' expression_list ')'
    {
        $$ = implied_do_list( $2, $4, $6 );
    }
;

```

```

format_statement:
    _RW_FORMAT
    {
        format_statement( $1 );
    }
;

```

```
%%
```

```
FILE: etimer/include/list.h
```

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```

#define LIST struct list_type
LIST
{
    char *identifier;
    char *type;
    int number;
    int length;
    char *dependent;
    LIST *next;
};

```

```

extern LIST *end_list( );
extern LIST *add_list( );
extern LIST *find_list( );
extern void print_list( );
extern void delete_list( );

```

```
FILE: etimer/library/Makefile
```

```

#
# Copyright 1991
# Georgia Institute of Technology
# Computer Engineering Research Laboratory
# Author: Stephen R. Wachtel
#

```

```

CC = cc -g
INCLUDE = ../include
CFLAGS = -IS(INCLUDE)
LIBRARY = library.a

```

```

OBJECTS = \
    alias.o \
    array.o \
    collapse.o \
    count.o \
    duplicate.o \
    hollerith.o \
    implied_do_list.o \
    label.o \
    link_list.o \
    list.o \
    lowercase.o \
    main.o \

```



```

margin_printf.o \
merge.o \
non_blank.o \
parse.o \
print_level.o \
stack.o \
statement.o \
summary.o \
timer.o \
type.o \
type_name.o \
update.o \
yyerror.o \
yygetc.o \
yywrap.o

$(LIBRARY): $(OBJECTS)
    rm -f $(LIBRARY)
    ar crv $(LIBRARY) $(OBJECTS)
    ranlib $(LIBRARY)

.SUFFIXES: .c .o
.c.o:
    $(CC) -c $(CFLAGS) $<

clean:
    rm -f $(LIBRARY) $(OBJECTS)

FILE: etimer/library/alias.c

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include <stdio.h>
#include <string.h>

extern int yylineno;
extern char *duplicate( );
extern char *lowercase( );

#define ALIAS struct alias_type
ALIAS
{
    char *old_identifier;
    char *new_identifier;
};

static ALIAS alias_table[ ] =
{
    { "", "" }
};

#define ALIAS_TABLE ( sizeof( alias_table ) / sizeof( ALIAS ) )

char *alias( identifier )
register char *identifier;
{
    register int low, high;
    register int middle, test;

    lowercase( identifier );

    low = 0;
    high = ALIAS_TABLE - 1;
    while ( low <= high )

```

```

{
    middle = ( low + high ) / 2;
    test = strcmp( identifier, alias_table[ middle ].old_identifier );

    if ( test < 0 )
    {
        high = middle - 1;
        continue;
    }

    if ( test > 0 )
    {
        low = middle + 1;
        continue;
    }

    fprintf( stderr, "line %d, %s aliased to %s\n", yylineno, identifier, alias_table[
middle ].new_identifier );
    return( alias_table[ middle ].new_identifier );
}

return( identifier );
} /* alias */

```

FILE: etimer/library/array.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

extern char *list( );
extern char *merge( );

char *array( identifier, optional_expression_list )
register char *identifier;
register char *optional_expression_list;
{
    if ( optional_expression_list != (char *)0 )
        return( merge( "%s(%s)", identifier, list( optional_expression_list, " , " ) ) );
    else
        return( merge( "%s()", identifier ) );
} /* array */

```

FILE: etimer/library/collapse.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include <stdio.h>
#include <string.h>

extern char *merge( );
extern char *parse( );

char *collapse( input_list )
register char *input_list;
{
    register char *input = (char *)NULL;
    int number = -1;

    register char *old_input = (char *)NULL;
    int old_number = -1;

    register char *output_list = (char *)NULL;
    register char *output;

```

```

while ( ( input = parse( input_list ) ) != (char *)NULL )
{
    sscanf( input, "%d", &number );
    if ( ( old_number + 1 ) != number )
    {
        if ( old_input != (char *)NULL )
        {
            if ( output_list != (char *)NULL )
            {
                output = merge( "%s{%s}{%s}", output_list, old_input, input );
                free( output_list );
            }
            else
                output = merge( "{%s}{%s}", old_input, input );
            free( old_input );
        }
        else
        {
            if ( output_list != (char *)NULL )
            {
                output = merge( "%s{%s}", output_list, input );
                free( output_list );
            }
            else
                output = merge( "{%s}", input );
        }

        free( input );
        output_list = output;
        old_input = (char *)NULL;
    }
    else
        old_input = input;
    old_number = number;
}

if ( old_input != (char *)NULL )
{
    if ( output_list != (char *)NULL )
    {
        output = merge( "%s{%s}", output_list, old_input );
        free( output_list );
    }
    else
        output = merge( "{%s}", old_input );

    free( old_input );
    output_list = output;
}

return( output_list );
} /* collapse */

```

FILE: etimer/library/count.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```

int count( *string, length, c )
register char *string;
register int length;
register char c;
{
    register int c_count = 0;
    while ( length != 0 )
    {
        if ( *string == c )
            c_count++;
    }
}

```

```

        string++;
        length--;
    }

    return( c_count );
} /* count */

```

FILE: etimer/library/duplicate.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include <stdio.h>
#include <string.h>
#include <malloc.h>

char *duplicate( string )
register char *string;
{
    register char *temporary = (char *)NULL;

    if ( string != (char *)NULL )
    {
        if ( ( temporary = (char *)malloc( strlen( string ) + 1 ) ) != (char *)NULL )
            strcpy( temporary, string );
        else
            fprintf( stderr, "ERROR: duplicate( %s )\n", string );
    }

    return( temporary );
} /* duplicate */

```

FILE: etimer/library/hollerith.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include <stdio.h>

char *hollerith( string, delimiter )
register char *string;
register char delimiter;
{
    int hollerith_length;
    register int string_length = 0;

    sscanf( string, "%dh", &hollerith_length );

    string[ string_length++ ] = delimiter;
    while ( hollerith_length != 0 )
    {
        if ( ( string[ string_length ] = yyinput( ) ) == '\n' )
        {
            yyinput( string[ string_length ] );
            break;
        }

        string_length++;
        hollerith_length--;
    }
    string[ string_length++ ] = delimiter;
    string[ string_length ] = '\0';
    return( string );
}

```

```

} /* hollerith */

```

```

FILE: etimer/library/IMPLIED_DO_LIST.C

```

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

extern char *list( );
extern char *merge( );

char *IMPLIED_DO_LIST( variable_list, identifier, expression_list )
register char *variable_list;
register char *identifier;
register char *expression_list;
{
    return( merge( "(%s, %s = %s)", list( variable_list, " ", " ), identifier, list(
expression_list, " ", " ) ) );
} /* IMPLIED_DO_LIST */

```

```

FILE: etimer/library/LABEL.C

```

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

extern int level;

char *label( string )
register char *string;
{
    if ( string != (char *)0 )
        margin_printf( "%d\t", atoi( string ) );
    else
        margin_printf( "\t" );

    while ( check_stack( string ) != 0 )
    {
        pull_stack( );
        level--;
    }

    return( string );
} /* label */

```

```

FILE: etimer/library/LINK_LIST.C

```

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include <stdio.h>
#include <malloc.h>
#include "list.h"

LIST *end_list( list )
register LIST *list;
{
    if ( list != (LIST *)NULL )
    {

```

```

        while ( list->next != (LIST *)NULL )
            list = list->next;
    }

    return( list );
} /* end_list */

LIST *add_list( list, identifier, type, number, length )
register LIST **list;
register char *identifier;
register char *type;
register int number;
register int length;
{
    register LIST *temporary = (LIST *)malloc( sizeof( LIST ) );

    temporary->identifier = identifier;
    temporary->type = type;
    temporary->number = number;
    temporary->length = length;
    temporary->dependent = (char *)NULL;
    temporary->next = (LIST *)NULL;

    if ( *list == (LIST *)NULL )
        *list = temporary;
    else
        end_list( *list )->next = temporary;

    return( temporary );
} /* add_list */

LIST *find_list( list, identifier, type )
register LIST *list;
register char *identifier;
register char *type;
{
    while ( list != (LIST *)NULL )
    {
        if ( ( strcmp( identifier, list->identifier ) == 0 )
            && ( strncmp( type, list->type, strlen( type ) ) == 0 ) )
            return( list );

        list = list->next;
    }

    fprintf( stderr, "ERROR: find_list( %s, %s )\n", identifier, type );
    exit( -1 );
} /* find_list */

void print_list( file, list )
register FILE *file;
register LIST *list;
{
    while ( list != (LIST *)NULL )
    {
        if ( list->identifier == (char *)NULL )
            fprintf( file, "\"\" " );
        else
            fprintf( file, "%s ", list->identifier );

        if ( list->type == (char *)NULL )
            fprintf( file, "\"\" " );
        else
            fprintf( file, "%s ", list->type );

        fprintf( file, "%d %d\n", list->number, list->length );

        list = list->next;
    }

    fprintf( file, "\n" );
} /* print_list */

void delete_list( list )
register LIST *list;
{
    if ( list != NULL )

```

```

    {
        delete_list( list->next );
        free( list );
    }
} /* delete_list */

```

FILE: etimer/library/list.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

extern char *parse( );
extern char *merge( );

char *list( input_list, delimiter )
register char *input_list;
register char *delimiter;
{
    register char *output_list;
    register char *list;
    register char *temporary;

    output_list = parse( input_list );
    list = parse( input_list );

    while ( list != (char *)0 )
    {
        temporary = merge( "%s%s%s", output_list, delimiter, list );

        free( output_list );
        free( list );

        output_list = temporary;
        list = parse( input_list );
    }

    return( output_list );
} /* list */

```

FILE: etimer/library/lowercase.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

char *lowercase( string )
register char *string;
{
    register int index = 0;

    while ( string[ index ] != '\0' )
    {
        string[ index ] = tolower( string[ index ] );
        index++;
    }

    return( string );
} /* lowercase */

```

FILE: etimer/library/main.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology

```

```

* Computer Engineering Research Laboratory
* Author: Stephen R. Wachtel
*/

```

```

#include <stdio.h>

```

```

extern FILE *yyin;
extern FILE *yyout;

```

```

#define PROGRAM argument[ 0 ]
#define INPUT_FILE argument[ 1 ]
#define OUTPUT_FILE argument[ 2 ]

```

```

int main( number_argument, argument )
int number_argument;
char *argument[ ];
{

```

```

    if ( number_argument == 1 )
    {
        yyin = stdin;
        yyout = stdout;

        yyparse( );
        exit( 0 );
    }

    if ( number_argument == 3 )
    {
        if ( ( yyin = fopen( INPUT_FILE, "r" ) ) == (FILE *)NULL )
        {
            fprintf( stderr, "%s: ERROR - unable to open input file '%s'\n", PROGRAM,
INPUT_FILE );
            exit( -1 );
        }

        if ( ( yyout = fopen( OUTPUT_FILE, "w" ) ) == (FILE *)NULL )
        {
            fprintf( stderr, "%s: ERROR - unable to open output file '%s'\n", PROGRAM,
OUTPUT_FILE );
            exit( -1 );
        }

        yyparse( );
        exit( 0 );
    }

    fprintf( stderr, "usage: %s <input file> <output file>\n", PROGRAM );
    exit( 0 );
} /* main */

```

```

FILE: etimer/library/margin_printf.c

```

```

/*
* Copyright 1991
* Georgia Institute of Technology
* Computer Engineering Research Laboratory
* Author: Stephen R. Wachtel
*/

```

```

#include <stdio.h>
#include <string.h>

```

```

extern FILE *yyin;
extern FILE *yyout;

```

```

static char buffer[ 256 * 20 ] = { 0 };

```

```

static void output_buffer( file )
register FILE *file;
{
#define LENGTH 72

```



```

int length = LENGTH;
int continuation = 0;
int quote = 0;
char temporary;

while ( strlen( buffer ) > length )
{
    if ( continuation++ != 0 )
        fprintf( file, "    &" );

    quote += count( buffer, length, '\\' );
    if ( ( quote % 2 ) == 0 )
    {
        while ( length != 0 )
        {
            if ( buffer[ length - 0 ] == '\\ ' )
                break;

            if ( buffer[ length - 0 ] == '\\,' )
                break;

            if ( buffer[ length - 1 ] == '\\' )
                break;

            length--;
        }

        if ( length == 0 )
        {
            fprintf( stderr, "ERROR: margin_printf()\n" );
            exit( -1 );
        }
    }

    temporary = buffer[ length ];
    buffer[ length ] = '\0';
    fprintf( file, "%s\n", buffer );
    buffer[ length ] = temporary;

    strcpy( &buffer[ 0 ], &buffer[ length ] );
    length = LENGTH - 6;
}

if ( strlen( buffer ) != 0 )
{
    if ( continuation++ != 0 )
        fprintf( file, "    &" );

    fprintf( file, "%s\n", buffer );
}
} /* output_buffer */

void margin_printf( format, a, b, c, d, e )
char *format;
int a, b, c, d, e;
{
    char temporary[ 256 * 20 ];

    sprintf( temporary, format, a, b, c, d, e );
    strcat( buffer, temporary );

    if ( buffer[ strlen( buffer ) - 1 ] == '\n' )
    {
        buffer[ strlen( buffer ) - 1 ] = '\0';

        while ( buffer[ strlen( buffer ) - 1 ] == ' ' )
            buffer[ strlen( buffer ) - 1 ] = '\0';

        switch ( buffer[ 0 ] )
        {
            case '\0':
                fprintf( yyout, "\n" );
                break;

            case '*':
            case 'c':
            case 'C':
                fprintf( yyout, "%s\n", buffer );
                break;
        }
    }
}

```

```

        default:
            output_buffer( yyout );
    }

    buffer[ 0 ] = '\0';
}
} /* margin_printf */

FILE: etimer/library/merge.c

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include <stdio.h>
#include <string.h>
#include <malloc.h>

#define STRLEN( s ) ( strlen( s ) - 2 )

char *merge( string, a, b, c, d )
register char *string;
register char *a;
register char *b;
register char *c;
register char *d;
{
    register char *temporary = (char *)NULL;

    switch ( count( string, strlen( string ), '%' ) )
    {
        case 0:
            if ( ( temporary = (char *)malloc( strlen( string ) + 1 ) ) != (char *)NULL )
                sprintf( temporary, string );
            else
                fprintf( stderr, "ERROR: merge( %s )\n", string );
            break;

        case 1:
            if ( ( temporary = (char *)malloc( strlen( string ) + STRLEN( a ) + 1 ) ) !=
(char *)NULL )
                sprintf( temporary, string, a );
            else
                fprintf( stderr, "ERROR: merge( %s, %s )\n", string, a );
            break;

        case 2:
            if ( ( temporary = (char *)malloc( strlen( string ) + STRLEN( a ) + STRLEN( b
) + 1 ) ) != (char *)NULL )
                sprintf( temporary, string, a, b );
            else
                fprintf( stderr, "ERROR: merge( %s, %s, %s )\n", string, a, b );
            break;

        case 3:
            if ( ( temporary = (char *)malloc( strlen( string ) + STRLEN( a ) + STRLEN( b
) + STRLEN( c ) + 1 ) ) != (char *)NULL )
                sprintf( temporary, string, a, b, c );
            else
                fprintf( stderr, "ERROR: merge( %s, %s, %s, %s )\n", string, a, b, c );
            break;

        case 4:
            if ( ( temporary = (char *)malloc( strlen( string ) + STRLEN( a ) + STRLEN( b
) + STRLEN( c ) + STRLEN( d ) + 1 ) ) != (char *)NULL )
                sprintf( temporary, string, a, b, c, d );
            else
                fprintf( stderr, "ERROR: merge( %s, %s, %s, %s, %s )\n", string, a, b, c, d
);
            break;

        default:
            fprintf( stderr, "ERROR: merge( %s )\n", string );
    }
}

```

```

        break;
    }

    return( temporary );
} /* merge */

```

FILE: etimer/library/non\_blank.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include <string.h>

char *non_blank( string )
register Char *string;
{
    register int offset;
    register int length;

    length = strlen( string ) - 1;
    while ( ( string[ length ] == ' ' ) && ( string[ length ] != '\0' ) )
        string[ length-- ] = '\0';

    offset = 0;
    while ( ( string[ offset ] == ' ' ) && ( string[ offset ] != '\0' ) )
        string[ offset++ ] = '\0';

    strcpy( string, &string[ offset ] );

    if ( strlen( string ) != 0 )
        return( string );
    else
        return( 0 );
} /* non_blank */

```

FILE: etimer/library/parse.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include <string.h>

extern char *duplicate( );

char *parse( list )
register char *list;
{
    register int length = 0;
    register int brace = 0;
    register char *temporary = (char *)0;

    for (;;)
    {
        switch ( list[ length ] )
        {
            case '(':
                brace++;
                break;

            case ')':
                brace--;
                break;
        }
        length++;
    }
}

```

```

        if ( brace == 0 )
            break;

        length++;
    }

    if ( length != 0 )
    {
        list[ length ] = '\0';
        temporary = duplicate( list + 1 );
        strcpy( list, list + 1 + length );
    }
    else
    {
        if ( list[ length ] != '\0' )
        {
            temporary = duplicate( list );
            list[ length ] = '\0';
        }
    }

    return( temporary );
} /* parse */

```

FILE: etimer/library/print\_level.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```
int level = 0;
```

```

void print_level( level )
register int level;
{
    if ( level != 0 )
    {
        while ( level-- != 0 )
            margin_printf( " " );
    }
} /* print_level */

```

FILE: etimer/library/stack.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```
extern int level;
```

```

#define STACK 128
static struct
{
    char *label;
} stack[ STACK ];
static int pointer = 0;

```

```

int push_stack( label )
register char *label;
{
    if ( pointer != STACK )
    {
        stack[ pointer ].label = label;

        pointer++;
        return( 1 );
    }
}

```

```

    }

    return( 0 );
} /* push_stack */

int check_stack( label )
register char *label;
{
    if ( pointer != 0 )
    {
        if ( strcmp( stack[ pointer - 1 ].label, label ) == 0 )
            return( 1 );
    }

    return( 0 );
} /* check_stack */

int pull_stack( )
{
    if ( pointer != 0 )
    {
        pointer--;

        free( stack[ pointer ].label );
        return( 1 );
    }

    return( 0 );
} /* pull_stack */

```

FILE: etimer/library/statement.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

void statement( label )
register char *label;
{
} /* statement */

```

FILE: etimer/library/summary.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include <stdio.h>
#include <string.h>
#include "list.h"

extern LIST *event_list;
extern int event_number;
extern char *collapse( );
extern char *duplicate( );
extern char *list( );
extern char *type_name( );
extern char *update( );

static void update_event_number( event )
register LIST *event;
{
    register LIST *temporary;

    while ( event != 0 )

```

```

{
    if ( event->number == 0 )
    {
        temporary = find_list( event_list, event->identifier, "send_" );
        event->number = temporary->number;
    }

    event = event->next;
}
} /* update_event_number */

static void update_dependent( event )
register LIST *event;
{
    register char *dependent = 0;
    char buffer[ 256 ];

    while ( event != 0 )
    {
        if ( event->identifier == 0 )
            event->identifier = duplicate( "?" );

        if ( strcmp( event->type, "program" ) == 0 )
        {
            dependent = 0;
        }

        if ( strcmp( event->type, "computation" ) == 0 )
        {
            event->dependent = dependent;

            sprintf( buffer, "%d", event->number );
            dependent = duplicate( buffer );
        }

        if ( strncmp( event->type, "send_", strlen( "send_" ) ) == 0 )
        {
            event->dependent = dependent;

            sprintf( buffer, "%d", event->number );
            dependent = duplicate( buffer );
        }

        if ( strncmp( event->type, "receive_", strlen( "receive_" ) ) == 0 )
        {
            sprintf( buffer, "%d", event->number );

            dependent = update( dependent, duplicate( buffer ) );
        }

        event = event->next;
    }
} /* update_dependent */

static void output_event_list( event )
register LIST *event;
{
    fprintf( stdout, "1 project project 1 \\\"\\\"\\n" );

    while ( event != 0 )
    {
        if ( event->dependent != 0 )
            event->dependent = list( collapse( event->dependent ), "," );

        if ( strcmp( event->type, "program" ) == 0 )
        {
            fprintf( stdout, "%d program %s 2 \\\"%s\\\"\\n", event->number, event->identifier,
event->dependent );
        }

        if ( strcmp( event->type, "computation" ) == 0 )
        {
            fprintf( stdout, "%d computation %s 3 \\\"%s\\\"\\n", event->number, list( event-
>identifier, "," ), event->dependent );
        }

        if ( strncmp( event->type, "send_", strlen( "send_" ) ) == 0 )
        {
            fprintf( stdout, "%d communication %s %d %s 3 \\\"%s\\\"\\n", event->number, event-

```

```

>identifier, event->length, type_name( &event->type[ 5 ] ), event->dependent );
    }

    if ( strcmp( event->type, "receive_", strlen( "receive_" ) ) == 0 )
    {
    }

    event = event->next;
}
} /* output_event_list */

```

```

void summary( )
{
#ifdef DEBUG
    fprintf( stderr, "pass 1:\n" );
    print_list( stderr, event_list );
#endif

    update_event_number( event_list );

#ifdef DEBUG
    fprintf( stderr, "pass 2:\n" );
    print_list( stderr, event_list );
#endif

    update_dependent( event_list );

#ifdef DEBUG
    fprintf( stderr, "pass 3:\n" );
    print_list( stderr, event_list );
#endif

    output_event_list( event_list );
} /* summary */

```

FILE: etimer/library/timer.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include <stdio.h>
#include <string.h>
#include "list.h"

extern int level;
extern char *update( );

LIST *event_list = 0;
int event_number = 1;

static int state = 0;

void program( identifier )
register char *identifier;
{
    add_list( &event_list, identifier, "program", ++event_number, 0 );

    state = 0;
} /* program */

int timer( comment )
register char *comment;
{
    if ( strcmp( comment, "**LOOP*", 6 ) != 0 )
        return( 0 );

    if ( strcmp( comment, "**LOOP* PROLOGUE\n" ) == 0 )
    {

```

```

    label( 0 );
    print_level( level );
    margin_printf( "CALL timer_prologue()\n" );

    state = 0;
    return( 1 );
}

if ( strcmp( comment, "**LOOP* START\n" ) == 0 )
{
    state = 1;
    return( 1 );
}

if ( strcmp( comment, "**LOOP* STOP\n" ) == 0 )
{
    if ( state == 2 )
    {
        label( 0 );
        print_level( level );
        margin_printf( "CALL stop_timer(%d)\n", event_number );
    }

    state = 0;
    return( 1 );
}

if ( strcmp( comment, "**LOOP* EPILOGUE\n" ) == 0 )
{
    label( 0 );
    print_level( level );
    margin_printf( "CALL timer_epilogue()\n" );

    state = 0;
    return( 1 );
}

return( 0 );
} /* timer */

void communication( type, identifier )
register char *type;
register char *identifier;
{
    register LIST *event = end_list( event_list );

    if ( state == 2 )
    {
        print_level( level );
        margin_printf( "CALL stop_timer(%d)\n", event_number );
        label( 0 );
    }

    state = 1;

    identifier[ strcspn( identifier, "()" ) ] = '\0';

    if ( strcmp( identifier, event->identifier ) == 0 )
    {
        event->length++;
        return;
    }

    if ( strncmp( type, "send_", strlen( "send_" ) ) == 0 )
        add_list( &event_list, identifier, type, ++event_number, 1 );

    if ( strncmp( type, "receive_", strlen( "receive_" ) ) == 0 )
        add_list( &event_list, identifier, type, 0, 1 );
} /* communication */

void computation( identifier )
register char *identifier;
{
    register LIST *event;

    switch ( state )
    {
        case 1:
            print_level( level );

```



```

        margin_printf( "CALL start_timer(%d)\n", ++event_number );
        label( 0 );

        add_list( &event_list, 0, "computation", event_number, 0 );

        case 2:
            if ( identifier != 0 )
            {
                event = end_list( event_list );
                event->identifier = update( event->identifier, identifier );
            }

            state = 2;
            break;
        }
    } /* computation */

```

FILE: etimer/library/type.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

extern char *merge( );

char *type( type_name, optional_type_length )
register char *type_name;
register char *optional_type_length;
{
    if ( optional_type_length != (char *)0 )
        return( merge( "%s%s", type_name, optional_type_length ) );
    else
        return( type_name );
} /* type */

```

FILE: etimer/library/type\_name.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include <stdio.h>
#include <string.h>

char *type_name( message_type )
register char *message_type;
{
    if ( strcmp( "character_08bit", message_type ) == 0 )
        return( "character*1" );

    if ( strcmp( "complex_32bit", message_type ) == 0 )
        return( "complex*8" );
    if ( strcmp( "complex_64bit", message_type ) == 0 )
        return( "complex*16" );

    if ( strcmp( "logical_08bit", message_type ) == 0 )
        return( "logical*1" );
    if ( strcmp( "logical_16bit", message_type ) == 0 )
        return( "logical*2" );
    if ( strcmp( "logical_32bit", message_type ) == 0 )
        return( "logical*4" );

    if ( strcmp( "real_32bit", message_type ) == 0 )
        return( "real*4" );
    if ( strcmp( "real_64bit", message_type ) == 0 )
        return( "real*8" );
}

```

```

    if ( strcmp( "signed_08bit", message_type ) == 0 )
        return( "integer*1" );
    if ( strcmp( "signed_16bit", message_type ) == 0 )
        return( "integer*2" );
    if ( strcmp( "signed_32bit", message_type ) == 0 )
        return( "integer*4" );

    if ( strcmp( "unsigned_08bit", message_type ) == 0 )
        return( "unsigned_integer*1" );
    if ( strcmp( "unsigned_16bit", message_type ) == 0 )
        return( "unsigned_integer*2" );
    if ( strcmp( "unsigned_32bit", message_type ) == 0 )
        return( "unsigned_integer*4" );

    fprintf( stderr, "ERROR: unrecognized message_type '%s'\n", message_type );
    exit( -1 );
} /* type_name */

```

FILE: etimer/library/update.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include <stdio.h>
#include <string.h>

extern char *merge( );
extern char *parse( );

char *update( input_list, identifier )
register char *input_list;
register char *identifier;
{
    register char *input;
    register char *output_list;
    register char *output;

    if ( identifier == (char *)NULL )
        return( input_list );

    output_list = (char *)NULL;

    while ( ( input = parse( input_list ) ) != (char *)NULL )
    {
        if ( strcmp( input, identifier ) == 0 )
            identifier = (char *)NULL;

        if ( output_list != (char *)NULL )
        {
            output = merge( "%s%s", output_list, input );
            free( output_list );
        }
        else
            output = merge( "{%s}", input );

        free( input );
        output_list = output;
    }

    if ( identifier == (char *)NULL )
        return( output_list );

    if ( output_list != (char *)NULL )
    {
        output = merge( "%s%s", output_list, identifier );
        free( output_list );
    }
    else
        output = merge( "{%s}", identifier );

    output_list = output;
}

```

```

    return( output_list );
} /* update */

```

FILE: etimer/library/yyerror.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```

#include <stdio.h>

```

```

extern int yylineno;

```

```

void yyerror( string )
register char *string;
{
    fprintf( stderr, "line %d, %s\n", yylineno, string );
    exit( -1 );
} /* yyerror */

```

FILE: etimer/library/yygetc.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```

#include <stdio.h>
#include <ctype.h>

```

```

extern int yylineno;

```

```

int tab( length )
register int length;
{
    while ( length-- != 0 )
        yyunput( ' ' );
    return( ' ' );
} /* tab */

```

```

int yygetc( file )
register FILE *file;
{
    int c;
    int column[ 6 ];

```

```

loop:
    if ( ( c = getc( file ) ) == '\t' )
        c = tab( 6 );

    if ( c != '\n' )
        return( c );

    if ( ( column[ 0 ] = getc( file ) ) != ' ' )
        goto abort_0;
    if ( ( column[ 1 ] = getc( file ) ) != ' ' )
        goto abort_1;
    if ( ( column[ 2 ] = getc( file ) ) != ' ' )
        goto abort_2;
    if ( ( column[ 3 ] = getc( file ) ) != ' ' )
        goto abort_3;
    if ( ( column[ 4 ] = getc( file ) ) != ' ' )
        goto abort_4;

```

```

    if ( isspace( column[ 5 ] =getc( file ) ) )
        goto abort_5;

    yylineno++;
    goto loop;

abort_5:
    if ( column[ 5 ] == '\t' )
        tab( 1 );
    else
    {
        yyunput( column[ 5 ] );
        if ( column[ 5 ] == '\n' )
            yylineno++;
    }

abort_4:
    if ( column[ 4 ] == '\t' )
        tab( 2 );
    else
    {
        yyunput( column[ 4 ] );
        if ( column[ 4 ] == '\n' )
            yylineno++;
    }

abort_3:
    if ( column[ 3 ] == '\t' )
        tab( 3 );
    else
    {
        yyunput( column[ 3 ] );
        if ( column[ 3 ] == '\n' )
            yylineno++;
    }

abort_2:
    if ( column[ 2 ] == '\t' )
        tab( 4 );
    else
    {
        yyunput( column[ 2 ] );
        if ( column[ 2 ] == '\n' )
            yylineno++;
    }

abort_1:
    if ( column[ 1 ] == '\t' )
        tab( 5 );
    else
    {
        yyunput( column[ 1 ] );
        if ( column[ 1 ] == '\n' )
            yylineno++;
    }

abort_0:
    if ( column[ 0 ] == '\t' )
        tab( 6 );
    else
    {
        yyunput( column[ 0 ] );
        if ( column[ 0 ] == '\n' )
            yylineno++;
    }

    return( c );
, /* yygetc */

```

FILE: etimer/library/yywrap.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```
int yywrap( )
{
    return( 1 );
} /* yywrap */
```

FILE: etimer/scanner.1

```
%(
/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */
%)
```

```
%a 10000
%e 10000
%k 10000
%n 10000
%o 10000
%p 10000
```

```
a  [aA]
b  [bB]
c  [cC]
d  [dD]
e  [eE]
f  [fF]
g  [gG]
h  [hH]
i  [iI]
j  [jJ]
k  [kK]
l  [lL]
m  [mM]
n  [nN]
o  [oO]
p  [pP]
q  [qQ]
r  [rR]
s  [sS]
t  [tT]
u  [uU]
v  [vV]
w  [wW]
x  [xX]
y  [yY]
z  [zZ]
```

```
%(
#include "grammar.h"
extern char *yylval;
```

```
#undef YYLMAX
#define YYLMAX (256*20)
```

```
extern char *duplicate( );
extern char *hollerith( );
extern char *non_blank( );
extern char *alias( );
%)
```

```
%%
```

```
^([\\*cC].*{\\n} )
^([\\ ])*{\\n} {
#ifdef DEBUG
    ECHO;
#endif
    yyval = duplicate( yytext );
    return( COMMENT );
```

```

    }

[\\] {
#ifdef DEBUG
    ECHO;
#endif
    /* return( '\\ ' ) */;
}

[\\&] {
#ifdef DEBUG
    ECHO;
#endif
    return( '\\&' );
}

[\\()] {
#ifdef DEBUG
    ECHO;
#endif
    return( '\\(' );
}

[\\)] {
#ifdef DEBUG
    ECHO;
#endif
    return( '\\)' );
}

[\\*] {
#ifdef DEBUG
    ECHO;
#endif
    return( '\\*' );
}

[\\*][\\*] {
#ifdef DEBUG
    ECHO;
#endif
    return( EXPONENTIATE );
}

[\\+] {
#ifdef DEBUG
    ECHO;
#endif
    return( '\\+' );
}

[\\,] {
#ifdef DEBUG
    ECHO;
#endif
    return( '\\,' );
}

[\\-] {
#ifdef DEBUG
    ECHO;
#endif
    return( '\\-' );
}

[\\.] {
#ifdef DEBUG
    ECHO;
#endif
    return( '\\.' );
}

```

```

[\\] {
#ifdef DEBUG
    ECHO;
#endif
    return( '\\' );
}

[\\:] {
#ifdef DEBUG
    ECHO;
#endif
    return( '\\:' );
}

[\\=] {
#ifdef DEBUG
    ECHO;
#endif
    return( '\\=' );
}

[\\n] {
#ifdef DEBUG
    ECHO;
#endif
    /* return( '\\n' ) */;
}

[\\t] {
#ifdef DEBUG
    ECHO;
#endif
    /* return( '\\t' ) */;
}

[\\.](a){n}(d)[\\.] {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_AND );
}

[\\.](e){q}[\\.] {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_EQ );
}

[\\.](e){q}(v)[\\.] {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_EQV );
}

[\\.](f){a}{l}{s}(e)[\\.] {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_FALSE );
}

[\\.](g){e}[\\.] {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_GE );
}

```

```

[\.]{g}{t}{\.} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_GT );
}

[\.]{l}{e}{\.} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_LE );
}

[\.]{l}{t}{\.} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_LT );
}

[\.]{n}{e}{\.} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_NE );
}

[\.]{n}{e}{q}{v}{\.} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_NEQV );
}

[\.]{n}{o}{t}{\.} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_NOT );
}

[\.]{o}{r}{\.} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_OR );
}

[\.]{t}{r}{u}{e}{\.} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_TRUE );
}

{a}{s}{s}{i}{g}{n} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_ASSIGN );
}

{b}{a}{c}{k}{s}{p}{a}{c}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_BACKSPACE );
}

```



```
{b}{l}{o}{c}{k}{\ }*(d){a}{t}{a} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_BLOCK_DATA );
}
```

```
{c}{a}{l}{l} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_CALL );
}
```

```
{c}{h}{a}{r}{a}{c}{t}{e}{r} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_CHARACTER );
}
```

```
{c}{l}{o}{s}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_CLOSE );
}
```

```
{c}{o}{m}{m}{o}{n} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_COMMON );
}
```

```
{c}{o}{m}{p}{l}{e}{x} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_COMPLEX );
}
```

```
{c}{o}{n}{t}{i}{n}{u}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_CONTINUE );
}
```

```
{d}{a}{t}{a} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_DATA );
}
```

```
{d}{i}{m}{e}{n}{s}{i}{o}{n} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_DIMENSION );
}
```

```
{d}{o} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_DO );
}
```

```
{d}{o}{u}{b}{l}{e}{\ }*(p){r}{e}{c}{i}{s}{i}{o}{n} {
```

```

#ifdef DEBUG
    ECHO;
#endif
    return( RW_DOUBLE_PRECISION );
}

```

```

{e}{l}{s}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_ELSE );
}

```

```

{e}{l}{s}{e}[\ ]*{i}{f} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_ELSE_IF );
}

```

```

{e}{n}{d} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_END );
}

```

```

{e}{n}{d}[\ ]*{i}{f} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_END_IF );
}

```

```

{e}{n}{d}{f}{i}{l}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_ENDFILE );
}

```

```

{e}{n}{t}{r}{y} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_ENTRY );
}

```

```

{e}{q}{u}{i}{v}{a}{l}{e}{n}{c}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_EQUIVALENCE );
}

```

```

{e}{x}{t}{e}{r}{n}{a}{l} {
#ifdef DE3UG
    ECHO;
#endif
    return( RW_EXTERNAL );
}

```

```

{f}{o}{r}{m}{a}{t}.* {
#ifdef DEBUG
    ECHO;
#endif
    yy1val = duplicate( yytext );
    return( RW_FORMAT );
}

```

```

{f}{u}{n}{c}{t}{i}{o}{n} {

```

```

#ifdef DEBUG
    ECHO;
#endif
    return( RW_FUNCTION );
}

{g}{o}{\ }*(t){o} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_GO_TO );
}

{i}{f} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_IF );
}

{i}{m}{p}{l}{i}{c}{i}{t} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_IMPLICIT );
}

{i}{n}{c}{l}{u}{d}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_INCLUDE );
}

{i}{n}{q}{u}{i}{r}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_INQUIRE );
}

{i}{n}{t}{e}{g}{e}{r} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_INTEGER );
}

{i}{n}{t}{r}{i}{n}{s}{i}{c} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_INTRINSIC );
}

{l}{o}{g}{i}{c}{a}{l} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_LOGICAL );
}

{n}{a}{m}{e}{l}{i}{s}{t} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_NAMELIST );
}

{o}{p}{e}{n} {
#ifdef DEBUG

```

```

    ECHO;
#endif
    return( RW_OPEN );
}

{p}{a}{r}{a}{m}{e}{t}{e}{r} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_PARAMETER );
}

{p}{a}{u}{s}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_PAUSE );
}

{p}{r}{i}{n}{t} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_PRINT );
}

{p}{r}{o}{g}{r}{a}{m} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_PROGRAM );
}

{r}{e}{a}{d} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_READ );
}

{r}{e}{a}{l} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_REAL );
}

{r}{e}{t}{u}{r}{n} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_RETURN );
}

{r}{e}{w}{i}{n}{d} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_REWIND );
}

{s}{a}{v}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_SAVE );
}

{s}{t}{o}{p} {
#ifdef DEBUG
    ECHO;

```

```

#endif
    return( RW_STOP );
}

{s}{u}{b}{r}{o}{u}{t}{i}{n}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_SUBROUTINE );
}

{t}{h}{e}{n} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_THEN );
}

{t}{o} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_TO );
}

{w}{r}{i}{t}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_WRITE );
}

{u}{n}{d}{e}{f}{i}{n}{e}{d} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_UNDEFINED );
}

[%a-zA-Z][_a-zA-Z0-9]* {
#ifdef DEBUG
    ECHO;
#endif
    yylval = duplicate( alias( yytext ) );
    return( IDENTIFIER );
}

^[0-9 ][0-9 ][0-9 ][0-9 ][0-9 ][\ ] {
#ifdef DEBUG
    ECHO;
#endif
    yylval = duplicate( non_blank( yytext ) );
    return( LABEL );
}

[0-9]+ |
[0-9]+/\.[a-zA-Z]+\.[ ] {
#ifdef DEBUG
    ECHO;
#endif
    yylval = duplicate( yytext );
    return( INTEGER );
}

[0-9]+\.[0-9]*([eE][\+\-]?[0-9]+)? |
[0-9]*\.[0-9]+([eE][\+\-]?[0-9]+)? |
[0-9]+([eE][\+\-]?[0-9]+)? {
#ifdef DEBUG
    ECHO;
#endif
    yylval = duplicate( yytext );
    return( REAL );
}

```

```

    }

[0-9]+\.[0-9]*([dD][\+|-]?[0-9]+)?    |
[0-9]*\.[0-9]+([dD][\+|-]?[0-9]+)?    |
[0-9]+([dD][\+|-]?[0-9]+)?{
#ifdef DEBUG
    ECHO;
#endif
    yylval = duplicate( yytext );
    return( DOUBLE_PRECISION );
}

\[^\']*\\' |
\[^\"]*\\\" {
#ifdef DEBUG
    ECHO;
#endif
    yytext[ 0 ] = '\\';
    yytext[ strlen( yytext ) - 1 ] = '\\';
    yylval = duplicate( yytext );
    return( STRING );
}

[0-9]+[hH] {
#ifdef DEBUG
    ECHO;
#endif
    yylval = duplicate( hollerith( yytext, '\\' ) );
    return( HOLLERITH );
}

```

FILE: etimer/statement/Makefile

```

#
# Copyright 1991
# Georgia Institute of Technology
# Computer Engineering Research Laboratory
# Author: Stephen R. Wachtel
#

```

```

CC = cc -g
INCLUDE = ../include
CFLAGS = -I$(INCLUDE)
LIBRARY = statement.a

```

```

OBJECTS = \
    arithmetic_if_statement.o \
    assign_statement.o \
    assigned_go_to_statement.o \
    assignment_statement.o \
    backspace_statement.o \
    block_data_statement.o \
    block_if_statement.o \
    call_statement.o \
    close_statement.o \
    comment_statement.o \
    common_statement.o \
    computed_go_to_statement.o \
    continue_statement.o \
    data_statement.o \
    declaration_statement.o \
    dimension_statement.o \
    do_statement.o \
    else_if_statement.o \
    else_statement.o \
    end_if_statement.o \
    end_statement.o \
    endfile_statement.o \
    entry_statement.o \
    equivalence_statement.o \
    external_statement.o \
    format_statement.o \
    function_statement.o \
    implicit_statement.o \

```

```

include_statement.o \
inquire_statement.o \
intrinsic_statement.o \
logical_if_statement.o \
namelist_statement.o \
open_statement.o \
parameter_statement.o \
pause_statement.o \
print_statement.o \
program_statement.o \
read_statement.o \
return_statement.o \
rewind_statement.o \
save_statement.o \
stop_statement.o \
subroutine_statement.o \
unconditional_go_to_statement.o \
write_statement.o

```

```

$(LIBRARY): $(OBJECTS)
rm -f $(LIBRARY)
ar crv $(LIBRARY) $(OBJECTS)
ranlib $(LIBRARY)

```

```

.SUFFIXES: .c .o
.c.o:
    $(CC) -c $(CFLAGS) $<

```

```

clean:
    rm -f $(LIBRARY) $(OBJECTS)

```

```

FILE: etimer/statement/arithmetic_if_statement.c

```

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```

extern int level;
extern char *list();

```

```

void arithmetic_if_statement( expression, label_list )
register char *expression;
register char *label_list;
{
    print_level( level );
    margin_printf( "IF (%s) %s\n", expression, list( label_list, ", " ) );
} /* arithmetic_if_statement */

```

```

FILE: etimer/statement/assign_statement.c

```

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```

extern int level;

```

```

void assign_statement( label, identifier )
register char *label;
register char *identifier;
{
    print_level( level );
    margin_printf( "ASSIGN %s TO %s\n", label, identifier );
} /* assign_statement */

```

FILE: etimer/statement/assigned\_go\_to\_statement.c

```
/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

extern int level;
extern char *list( );

void assigned_go_to_statement( identifier, optional_label_list )
register char *identifier;
register char *optional_label_list;
{
    if ( optional_label_list != 0 )
    {
        print_level( level );
        margin_printf( "GO TO %s, (%s)\n", identifier, list( optional_label_list, " " ) );
    }
    else
    {
        print_level( level );
        margin_printf( "GO TO %s\n", identifier );
    }
} /* assigned_go_to_statement */
```

FILE: etimer/statement/assignment\_statement.c

```
/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

extern int level;

void assignment_statement( variable, expression )
register char *variable;
register char *expression;
{
    print_level( level );
    margin_printf( "%s = %s\n", variable, expression );
} /* assignment_statement */
```

FILE: etimer/statement/backspace\_statement.c

```
/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

extern int level;
extern char *list( );

void backspace_statement( control_list )
register char *control_list;
{
    print_level( level );
    margin_printf( "BACKSPACE (%s, %s), list (%s)\n", list( control_list, " " ) );
} /* backspace_statement */
```

FILE: etimer/statement/backspace\_data\_statement.c



```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```
extern int level;
```

```

void block_data_statement( identifier )
register char *identifier;
{
    print_level( level );
    margin_printf( "BLOCK DATA %s\n", identifier );
} /* block_data_statement */

```

```
FILE: etimer/statement/block_if_statement.c
```

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```
extern int level;
```

```

void block_if_statement( expression )
register char *expression;
{
    print_level( level );
    margin_printf( "IF (%s) THEN\n", expression );

    level++;
} /* block_if_statement */

```

```
FILE: etimer/statement/call_statement.c
```

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```

extern int level;
extern char *list();

```

```

void call_statement( identifier, optional_actual_argument_list )
register char *identifier;
register char *optional_actual_argument_list;
{
    if ( optional_actual_argument_list != 0 )
    {
        print_level( level );
        margin_printf( "CALL %s(%s)\n", identifier, list( optional_actual_argument_list,
            " " ) );
    }
    else
    {
        print_level( level );
        margin_printf( "CALL %s()\n", identifier );
    }
} /* call_statement */

```

```
FILE: etimer/statement/close_statement.c
```

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```

extern int level;
extern char *list( );

```

```

void close_statement( control_list )
register char *control_list;
{
    print_level( level );
    margin_printf( "CLOSE (%s)\n", list( control_list, " " ) );
} /* close_statement */

```

FILE: etimer/statement/comment\_statement.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```

void comment_statement( string )
register char *string;
{
    margin_printf( "%s", string );
} /* comment_statement */

```

FILE: etimer/statement/common\_statement.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```

extern int level;
extern char *parse( );
extern char *list( );

```

```

void common_statement( optional_common_name, common_list )
register char *optional_common_name;
register char *common_list;
{
    register char *common;
    register char *identifier;
    register char *optional_subscript_list;

    print_level( level );
    margin_printf( "COMMON " );

    if ( optional_common_name != 0 )
        margin_printf( "/%s/ ", optional_common_name );

    while ( common = parse( common_list ) )
    {
        identifier = parse( common );
        optional_subscript_list = parse( common );

        margin_printf( "%s", identifier );
        if ( optional_subscript_list != 0 )
            margin_printf( "({%s})", list( optional_subscript_list, " " ) );

        if ( strlen( common_list ) != 0 )
            margin_printf( ", " );
    }
}

```

```

        margin_printf( "\n" );
    } /* common_statement */

```

FILE: etimer/statement/computed\_go\_to\_statement.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

extern int level;
extern char *list( );

void computed_go_to_statement( label_list, expression )
register char *label_list;
register char *expression;
{
    print_level( level );
    margin_printf( "GO TO (%s), %s\n", list( label_list, " ", " ), expression );
} /* computed_go_to_statement */

```

FILE: etimer/statement/continue\_statement.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

extern int level;

void continue_statement( )
{
    print_level( level );
    margin_printf( "CONTINUE\n" );
} /* continue_statement */

```

FILE: etimer/statement/data\_statement.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

extern int level;
extern char *parse( );
extern char *list( );

void data_statement( data_list )
register char *data_list;
{
    register char *data;
    register char *variable_list;
    register char *constant_list;

    print_level( level );
    margin_printf( "DATA " );

    while ( data = parse( data_list ) )
    {
        variable_list = parse( data );
        constant_list = parse( data );

        margin_printf( "%s /%s/", list( variable_list, " ", " ), list( constant_list, " ", " )

```

```

);

    if ( strlen( data_list ) != 0 )
        margin_printf( " ", " );
    }

    margin_printf( "\n" );
} /* data_statement */

```

FILE: etimer/statement/declaration\_statement.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

extern int level;
extern char *parse( );
extern char *list( );

void declaration_statement( type, declaration_list )
register char *type;
register char *declaration_list;
{
    register char *declaration;
    register char *identifier;
    register char *optional_subscript_list;

    print_level( level );
    margin_printf( "%s ", type );

    while ( declaration = parse( declaration_list ) )
    {
        identifier = parse( declaration );
        optional_subscript_list = parse( declaration );

        margin_printf( "%s", identifier );
        if ( optional_subscript_list != 0 )
            margin_printf( "({%s}", list( optional_subscript_list, " ", " ) );

        if ( strlen( declaration_list ) != 0 )
            margin_printf( " ", " );

    }

    margin_printf( "\n" );
} /* declaration_statement */

```

FILE: etimer/statement/dimension\_statement.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

extern int level;
extern char *parse( );
extern char *list( );

void dimension_statement( dimension_list )
register char *dimension_list;
{
    register char *dimension;
    register char *identifier;
    register char *subscript_list;

    print_level( level );
    margin_printf( "DIMENSION " );

    while ( dimension = parse( dimension_list ) )

```

```

    {
        identifier = parse( dimension );
        subscript_list = parse( dimension );

        margin_printf( "%s(%s)", identifier, list( subscript_list, ", " ) );

        if ( strlen( dimension_list ) != 0 )
            margin_printf( ", " );
    }

    margin_printf( "\n" );
} /* dimension_statement */

```

FILE: etimer/statement/do\_statement.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

extern int level;
extern char *list( );

void do_statement( label, identifier, expression_list )
register char *label;
register char *identifier;
register char *expression_list;
{
    push_stack( label );

    print_level( level );
    margin_printf( "DO %s %s = %s\n", label, identifier, list( expression_list, ", " ) );

    level++;
} /* do_statement */

```

FILE: etimer/statement/else\_if\_statement.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

extern int level;

void else_if_statement( expression )
register char *expression;
{
    level--;

    print_level( level );
    margin_printf( "ELSE IF (%s) THEN\n", expression );

    level++;
} /* else_if_statement */

```

FILE: etimer/statement/else\_statement.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

extern int level;

```

```

void else_statement( )
{
    level--;

    print_level( level );
    margin_printf( "ELSE\n" );

    level++;
} /* else_statement */

```

FILE: etimer/statement/end\_if\_statement.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```
extern int level;
```

```

void end_if_statement( )
{
    level--;

    print_level( level );
    margin_printf( "END IF\n" );
} /* end_if_statement */

```

FILE: etimer/statement/end\_statement.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```
extern int level;
```

```

void end_statement( )
{
    print_level( level );
    margin_printf( "END\n" );
} /* end_statement */

```

FILE: etimer/statement/endfile\_statement.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```
extern int level;
extern char *list( );
```

```

void endfile_statement( control_list )
register char *control_list;
{
    print_level( level );
    margin_printf( "ENDFILE (%s)\n", list( control_list, ", " ) );
} /* endfile_statement */

```

FILE: etimer/statement/entry\_statement.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

extern int level;
extern char *list( );

void entry_statement( identifier, optional_formal_argument_list )
register char *identifier;
register char *optional_formal_argument_list;
{
    if ( optional_formal_argument_list != 0 )
    {
        print_level( level );
        margin_printf( "ENTRY %s(%s)\n", identifier, list( optional_formal_argument_list,
", " ) );
    }
    else
    {
        print_level( level );
        margin_printf( "ENTRY %s()\n", identifier );
    }
} /* entry_statement */

```

FILE: etimer/statement/equivalence\_statement.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

extern int level;
extern char *parse( );
extern char *list( );

void equivalence_statement( equivalence_list )
register char *equivalence_list;
{
    register char *variable_list;

    print_level( level );
    margin_printf( "EQUIVALENCE " );

    while ( variable_list = parse( equivalence_list ) )
    {
        margin_printf( "(%s)", list( variable_list, ", " ) );

        if ( strlen( equivalence_list ) != 0 )
            margin_printf( ", " );
    }

    margin_printf( "\n" );
} /* equivalence_statement */

```

FILE: etimer/statement/external\_statement.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

extern int level;
extern char *list( );

```

```

void external_statement( external_list )
register char *external_list;
{
    print_level( level );
    margin_printf( "EXTERNAL %s\n", list( external_list, ", " ) );
} /* external_statement */

```

FILE: etimer/statement/format\_statement.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```
extern int level;
```

```

void format_statement( format )
register char *format;
{
    format[0] = 'F';
    format[1] = 'O';
    format[2] = 'R';
    format[3] = 'M';
    format[4] = 'A';
    format[5] = 'T';

    print_level( level );
    margin_printf( "%s\n", format );
} /* format_statement */

```

FILE: etimer/statement/function\_statement.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```
extern int level;
extern char *list();
```

```

void function_statement( optional_type, identifier, optional_formal_argument_list )
register char *optional_type;
register char *identifier;
register char *optional_formal_argument_list;
{
    print_level( level );
    if ( optional_type != 0 )
        margin_printf( "%s ", optional_type );

    if ( optional_formal_argument_list != 0 )
        margin_printf( "FUNCTION %s(%s)\n", identifier, list(
optional_formal_argument_list, ", " ) );
    else
        margin_printf( "FUNCTION %s()\n", identifier );
} /* function_statement */

```

FILE: etimer/statement/implicit\_statement.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```



```
extern int level;
extern char *list( );
```

```
void implicit_statement( type, implicit_list )
register char *type;
register char *implicit_list;
{
    print_level( level );
    margin_printf( "IMPLICIT %s(%s)\n", type, list( implicit_list, ", " ) );
} /* implicit_statement */
```

FILE: etimer/statement/include\_statement.c

```
/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */
```

```
extern int level;
```

```
void include_statement( filename )
register char *filename;
{
    print_level( level );
    margin_printf( "INCLUDE %s\n", filename );
} /* include_statement */
```

FILE: etimer/statement/inquire\_statement.c

```
/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */
```

```
extern int level;
```

```
void inquire_statement( control_list )
register char *control_list;
{
    print_level( level );
    margin_printf( "INQUIRE (%s)\n", list( control_list, ", " ) );
} /* inquire_statement */
```

FILE: etimer/statement/intrinsic\_statement.c

```
/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */
```

```
extern int level;
extern char *list( );
```

```
void intrinsic_statement( intrinsic_list )
register char *intrinsic_list;
{
    print_level( level );
    margin_printf( "INTRINSIC %s\n", list( intrinsic_list, ", " ) );
} /* intrinsic_statement */
```

FILE: etimer/statement/logical\_if\_statement.c

```
/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

extern int level;
static int save_level;

void logical_if_statement( )
{
    level = save_level;
    save_level = 0;
} /* logical_if_statement */

void if_expression( expression )
register char *expression;
{
    print_level( level );
    margin_printf( "IF (%s) ", expression );

    save_level = level;
    level = 0;
} /* if_expression */
```

FILE: etimer/statement/namelist\_statement.c

```
/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

extern int level;
extern char *list( );

void namelist_statement( namelist_name, namelist_list )
register char *namelist_name;
register char *namelist_list;
{
    print_level( level );
    margin_printf( "NAMELIST %s/ %s\n", namelist_name, list( namelist_list, ", " ) );
} /* namelist_statement */
```

FILE: etimer/statement/open\_statement.c

```
/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

extern int level;
extern char *list( );

void open_statement( control_list )
register char *control_list;
{
    print_level( level );
    margin_printf( "OPEN (%s)\n", list( control_list, ", " ) );
} /* open_statement */
```

FILE: etimer/statement/parameter\_statement.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

extern int level;
extern char *parse();

void parameter_statement( parameter_list )
register char *parameter_list;
{
    register char *parameter;
    register char *identifier;
    register char *expression;

    print_level( level );
    margin_printf( "PARAMETER (" );

    while ( parameter = parse( parameter_list ) )
    {
        identifier = parse( parameter );
        expression = parse( parameter );

        margin_printf( "%s = %s", identifier, expression );

        if ( strlen( parameter_list ) != 0 )
            margin_printf( ", " );
    }

    margin_printf( ")\n" );
} /* parameter_statement */

```

FILE: etimer/statement/pause\_statement.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

extern int level;

void pause_statement( optional_expression )
register char *optional_expression;
{
    if ( optional_expression != 0 )
    {
        print_level( level );
        margin_printf( "PAUSE %s\n", optional_expression );
    }
    else
    {
        print_level( level );
        margin_printf( "PAUSE\n" );
    }
} /* pause_statement */

```

FILE: etimer/statement/print\_statement.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

extern int level;

```

```

extern char *list( );

void print_statement( control_list, optional_io_list )
register char *control_list;
register char *optional_io_list;
{
    if ( optional_io_list != 0 )
    {
        print_level( level );
        margin_printf( "PRINT (%s) %s\n", list( control_list, ", " ), list(
optional_io_list, ", " ) );
    }
    else
    {
        print_level( level );
        margin_printf( "PRINT (%s)\n", list( control_list, ", " ) );
    }
} /* print_statement */

```

FILE: etimer/statement/program\_statement.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```
extern int level;
```

```

void program_statement( identifier )
register char *identifier;
{
    print_level( level );
    margin_printf( "PROGRAM %s\n", identifier );
} /* program_statement */

```

FILE: etimer/statement/read\_statement.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```
extern int level;
extern char *list( );
```

```

void read_statement( control_list, optional_io_list )
register char *control_list;
register char *optional_io_list;
{
    if ( optional_io_list != 0 )
    {
        print_level( level );
        margin_printf( "READ (%s) %s\n", list( control_list, ", " ), list(
optional_io_list, ", " ) );
    }
    else
    {
        print_level( level );
        margin_printf( "READ (%s)\n", list( control_list, ", " ) );
    }
} /* read_statement */

```

FILE: etimer/statement/return\_statement.c

```

/*
 * Copyright 1991

```

```

* Georgia Institute of Technology
* Computer Engineering Research Laboratory
* Author: Stephen R. Wachtel
*/

```

```
extern int level;
```

```

void return_statement( expression )
register char *expression;
{
    if ( expression != 0 )
    {
        print_level( level );
        margin_printf( "RETURN %s\n", expression );
    }
    else
    {
        print_level( level );
        margin_printf( "RETURN\n" );
    }
} /* return_statement */

```

```
FILE: etimer/statement/rewind_statement.c
```

```

/*
* Copyright 1991
* Georgia Institute of Technology
* Computer Engineering Research Laboratory
* Author: Stephen R. Wachtel
*/

```

```

extern int level;
extern char *list();

```

```

void rewind_statement( control_list )
register char *control_list;
{
    print_level( level );
    margin_printf( "REWIND (%s)\n", list( control_list, " " ) );
} /* rewind_statement */

```

```
FILE: etimer/statement/save_statement.c
```

```

/*
* Copyright 1991
* Georgia Institute of Technology
* Computer Engineering Research Laboratory
* Author: Stephen R. Wachtel
*/

```

```

extern int level;
extern char *list();

```

```

void save_statement( save_list )
register char *save_list;
{
    print_level( level );
    margin_printf( "SAVE %s\n", list( save_list, " " ) );
} /* save_statement */

```

```
FILE: etimer/statement/stop_statement.c
```

```

/*
* Copyright 1991
* Georgia Institute of Technology
* Computer Engineering Research Laboratory
* Author: Stephen R. Wachtel
*/

```

```
extern int level;

void stop_statement( optional_expression )
register char *optional_expression;
{
    if ( optional_expression != 0 )
    {
        print_level( level );
        margin_printf( "STOP %s\n", optional_expression );
    }
    else
    {
        print_level( level );
        margin_printf( "STOP\n" );
    }
} /* stop_statement */
```

FILE: etimer/statement/subroutine\_statement.c

```
/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

extern int level;
extern char *list( );

void subroutine_statement( identifier, optional_formal_argument_list )
register char *identifier;
register char *optional_formal_argument_list;
{
    if ( optional_formal_argument_list != 0 )
    {
        print_level( level );
        margin_printf( "SUBROUTINE %s(%s)\n", identifier, list(
optional_formal_argument_list, ", " ) );
    }
    else
    {
        print_level( level );
        margin_printf( "SUBROUTINE %s()\n", identifier );
    }
} /* subroutine_statement */
```

FILE: etimer/statement/unconditional\_go\_to\_statement.c

```
/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

extern int level;

void unconditional_go_to_statement( label )
register char *label;
{
    print_level( level );
    margin_printf( "GO TO %s\n", label );
} /* unconditional_go_to_statement */
```

FILE: etimer/statement/write\_statement.c

```
/*
 * Copyright 1991
 * Georgia Institute of Technology
```

```
* Computer Engineering Research Laboratory  
* Author: Stephen R. Wachtel  
*/
```

```
extern int level;  
extern char *list( );
```

```
void write_statement( control_list, optional_io_list )  
register char *control_list;  
register char *optional_io_list;  
{  
    if ( optional_io_list != 0 )  
    {  
        print_level( level );  
        margin_printf( "WRITE (%s) %s\n", list( control_list, " " ), list(  
optional_io_list, " " ) );  
    }  
    else  
    {  
        print_level( level );  
        margin_printf( "WRITE (%s)\n", list( control_list, " " ) );  
    }  
} /* write_statement */
```

## 14. Appendix I: initial program source

FILE: initial/Makefile

```
#
# Copyright 1991
# Georgia Institute of Technology
# Computer Engineering Research Laboratory
# Author: Stephen R. Wachtel
#

default:    initial

CC = cc -g
INCLUDE = include
CFLAGS = -I$(INCLUDE)
LIBRARY = library/library.a

OBJECTS = \
    $(INCLUDE)/grammar.h \
    *grammar.[co] \
    *scanner.[co] \
    yytrace.[co] \
    y.output

PROGRAMS = \
    *initial

grammar.c: grammar.y
    yacc -dv grammar.y
    mv y.tab.h $(INCLUDE)/grammar.h
    mv y.tab.c grammar.c

scanner.c: scanner.l
    lex -vt scanner.l | sed 's/getc/yygetc/' >scanner.c

scanner.o: scanner.c $(INCLUDE)/grammar.h
    $(CC) $(CFLAGS) -c scanner.c

grammar.o: grammar.c
    $(CC) $(CFLAGS) -c grammar.c

initial:    grammar.o scanner.o $(LIBRARY)
    $(CC) -o initial grammar.o scanner.o $(LIBRARY)

sgrammar.c: grammar.c yytoken.awk
    awk -f yytoken.awk <grammar.c >sgrammar.c

sgrammar.o: sgrammar.c
    $(CC) $(CFLAGS) -c sgrammar.c

sinitial:    sgrammar.o $(LIBRARY)
    $(CC) -o sinitial sgrammar.o $(LIBRARY)

dscanner.c: scanner.c
    cp scanner.c dscanner.c

dscanner.o: dscanner.c $(INCLUDE)/grammar.h
    $(CC) $(CFLAGS) -DDEBUG -c dscanner.c

dinitial:    grammar.o dscanner.o $(LIBRARY)
    $(CC) -o dinitial grammar.o dscanner.o $(LIBRARY)

tgrammar.c: grammar.c
    sed 's/yystack:/& yytrace(yystate);/' <grammar.c >tgrammar.c

tgrammar.o: tgrammar.c
    $(CC) $(CFLAGS) -c tgrammar.c
```



```
tinitial: tgrammar.o scanner.o yytrace.o $(LIBRARY)
          $(CC) -o tinitial tgrammar.o scanner.o yytrace.o $(LIBRARY)
```

```
yytrace.c: grammar.c yytrace.awk
          awk -f yytrace.awk <y.output >yytrace.c
```

```
yytrace.o: yytrace.c
          $(CC) $(CFLAGS) -c yytrace.c
```

```
clean:
  cd library; make clean
  rm -f $(PROGRAMS) $(OBJECTS)
```

```
FILE: initial/grammar.y
```

```
/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */
```

```
/*
 * FORTRAN 77
 */
```

```
%token RW_AND
%token RW_ASSIGN
%token RW_BACKSPACE
%token RW_BLOCK_DATA
%token RW_CALL
%token RW_CHARACTERP
%token RW_CLOSE
%token RW_COMMON
%token RW_COMPLEX
%token RW_CONTINUE
%token RW_DATA
%token RW_DIMENSION
%token RW_DO
%token RW_DOUBLE_PRECISION
%token RW_ELSE
%token RW_ELSE_IF
%token RW_END
%token RW_END IF
%token RW_ENDFILE
%token RW_ENTRY
%token RW_EQ
%token RW_EQUIVALENCE
%token RW_EQV
%token RW_EXTERNAL
%token RW_FALSE
%token RW_FOKMAT
%token RW_FUNCTION
%token RW_GE
%token RW_GO_TO
%token RW_GT
%token RW_IF
%token RW_IMPLICIT
%token RW_INCLUDE
%token RW_INQUIRE
%token RW_INTEGER
%token RW_INTRINSIC
%token RW_LE
%token RW_LOGICAL
%token RW_LT
%token RW_NAMelist
%token RW_NE
%token RW_NEQV
%token RW_NOT
%token RW_OPEN
%token RW_OR
%token RW_PARAMETER
%token RW_PAUSE
%token RW_PRINT
%token RW_PROGRAM
```

```

%token RW_READ
%token RW_REAL
%token RW_RETURN
%token RW_REWIND
%token RW_SAVE
%token RW_STOP
%token RW_SUBROUTINE
%token RW_THEN
%token RW_TO
%token RW_TRUE
%token RW_WRITE
%token RW_UNDEFINED

%token CONCATENATE
%token COMMENT
%token DOUBLE_PRECISION
%token EXPONENTIATE
%token HOLLERITH
%token IDENTIFIER
%token INTEGER
%token LABEL
%token REAL
%token STRING

%left ','
%nonassoc ':'
%right '='
%left RW_EQV RW_NEQV
%left RW_OR
%left RW_AND
%left RW_NOT
%nonassoc RW_EQ RW_NE RW_LT RW_LE RW_GT RW_GE
%left CONCATENATE
%left '+' '-'
%left '*' '/'
%right EXPONENTIATE
%left SIGN

%{
typedef char *POINTER;
#define YYSTYPE POINTER

#include "usage.h"
static int usage = REF;
static int level = 0;

int conditional = 0;
int argument_number = 0;
%}

%%

program:
    optional_statement_list
    {
        summary( );
    }
    ;

optional_statement_list:
    /* NULL */
    ;
    statement_list
    ;

statement_list:
    statement
    {
        statement_list statement
    }
    ;

statement:
    comment_statement

```

```

|
|   label_unlabeled_statement
|
;

comment_statement:
    COMMENT
;

label:
    LABEL
;

unlabeled_statement:
    include_statement
    |
    program_statement
    |
    block_data_statement
    |
    function_statement
    |
    subroutine_statement
    |
    entry_statement
    |
    end_statement
    |
    specification_statement
    |
    { add_statement_list( 0, ( level != 0 ? conditional : 0 ) ); }
executable_statement
    |
    format_statement
;

include_statement:
    RW_INCLUDE character_constant
;

program_statement:
    RW_PROGRAM program_identifier
;

program_identifier:
    IDENTIFIER
    {
        begin_block( $1 );
    }
;

block_data_statement:
    RW_BLOCK_DATA block_data_identifier
;

block_data_identifier:
    IDENTIFIER
    {
        begin_block( $1 );
    }
;

function_statement:
    RW_FUNCTION function_identifier optional_formal_argument_list
    |
    type RW_FUNCTION function_identifier optional_formal_argument_list
;

function_identifier:
    IDENTIFIER
    {
        begin_block( $1 );
    }

```

```

    }
;

subroutine_statement:
    RW_SUBROUTINE subroutine_identifier
    |
    RW_SUBROUTINE subroutine_identifier optional_formal_argument_list
;

subroutine_identifier:
    IDENTIFIER
    {
        begin_block( $1 );
    }
;

entry_statement:
    RW_ENTRY entry_identifier
    |
    RW_ENTRY entry_identifier optional_formal_argument_list
;

entry_identifier:
    IDENTIFIER
;

optional_formal_argument_list:
    '(' ' ' ')'
    |
    '(' formal_argument_list ')'
;

formal_argument_list:
    formal_argument
    |
    formal_argument_list ',' formal_argument
;

formal_argument:
    IDENTIFIER
    {
        add_formal_argument_list( $1 );
    }
    |
    formal_argument_altername_return
    {
        add_formal_argument_list( $1 );
    }
;

formal_argument_altername_return:
    '*'
    {
        $$ = "";
    }
;

end_statement:
    RW_END
    {
        end_block( );
    }
;

specification_statement:
    external_statement
    |
    intrinsic_statement
    |
    { add_statement_list( 0, 0 ); } parameter_statement

```

```

dimension_statement
|
declaration_statement
|
save_statement
|
common_statement
|
equivalence_statement
|
implicit_statement
|
{ add_statement_list( 0, 0 ); } data_statement
|
namelist_statement
;

external_statement:
    RW_EXTERNAL external_list
;

external_list:
    external
|
    external_list ',' external
;

external:
    IDENTIFIER
;

intrinsic_statement:
    RW_INTRINSIC intrinsic_list
;

intrinsic_list:
    intrinsic
|
    intrinsic_list ',' intrinsic
;

intrinsic:
    IDENTIFIER
;

parameter_statement:
    RW_PARAMETER '(' parameter_list ')'
;

parameter_list:
    parameter
|
    parameter_list ',' parameter
;

parameter:
    parameter_identifier '=' expression
;

parameter_identifier:
    IDENTIFIER
|
    { add_identifier_list( $1, 0, SET ); }
;

dimension_statement:
    RW_DIMENSION dimension_list
;

```

```

dimension_list:
    dimension
    |
    dimension_list ',' dimension
    ;

dimension:
    IDENTIFIER '(' subscript_list ')'
    ;

subscript_list:
    subscript
    |
    subscript_list ',' subscript
    ;

subscript:
    upper_bound
    |
    lower_bound ':' upper_bound
    ;

lower_bound:
    INTEGER
    |
    '+' INTEGER
    |
    '-' INTEGER
    |
    IDENTIFIER
    |
    '+' IDENTIFIER
    |
    '-' IDENTIFIER
    |
    INTEGER '*' IDENTIFIER
    |
    IDENTIFIER '*' INTEGER
    ;

upper_bound:
    lower_bound
    |
    upper_bound_adjustable
    ;

upper_bound_adjustable:
    '*'
    ;

declaration_statement:
    type declaration_list
    ;

declaration_list:
    declaration
    |
    declaration_list ',' declaration
    ;

declaration:
    IDENTIFIER optional_type_length
    |
    IDENTIFIER '(' subscript_list ')' optional_type_length
    ;

type:
    type_name optional_type_length
    ;

```

```

type_name:
    RW_CHARACTER
    |
    RW_COMPLEX
    |
    RW_DOUBLE_PRECISION
    |
    RW_INTEGER
    |
    RW_LOGICAL
    |
    RW_REAL
    |
    RW_UNDEFINED
    ;

optional_type_length:
    /* NULL */
    |
    type_length
    ;

type_length:
    '*' INTEGER
    |
    '*' type_length_adjustable
    ;

type_length_adjustable:
    '(' '*' ')'
    ;

save_statement:
    RW_SAVE optional_save_list
    ;

optional_save_list:
    /* NULL */
    |
    save_list
    ;

save_list:
    save
    |
    save_list ',' save
    ;

save:
    IDENTIFIER
    |
    common_name
    ;

common_statement:
    RW_COMMON optional_common_name common_variable_list
    ;

optional_common_name:
    /* NULL */
    |
    common_name
    ;

common_name:
    '*' optional_identifier '/'
    ;

optional_identifier:

```

```

/* NULL */
IDENTIFIER
;

common_variable_list:
    common_variable
    ;
    common_variable_list ',' common_variable
;

common_variable:
    IDENTIFIER
    ;
    IDENTIFIER '(' subscript_list ')'
;

equivalence_statement:
    RW_EQUIVALENCE equivalence_list
;

equivalence_list:
    equivalence
    ;
    equivalence_list ',' equivalence
;

equivalence:
    '(' equivalence_variable_list ')'
;

equivalence_variable_list:
    equivalence_variable
    ;
    equivalence_variable_list ',' equivalence_variable
;

equivalence_variable:
    IDENTIFIER
    ;
    IDENTIFIER '(' subscript_list ')'
;

implicit_statement:
    RW_IMPLICIT type '(' implicit_list ')'
;

implicit_list:
    implicit
    ;
    implicit_list ',' implicit
;

implicit:
    IDENTIFIER
    ;
    IDENTIFIER '-' IDENTIFIER
;

namelist_statement:
    RW_NAMELIST namelist_name namelist_list
;

namelist_name:
    '/' IDENTIFIER '/'
;

namelist_list:

```



```

namelist
    namelist_list ',' namelist
;

namelist:
    IDENTIFIER
;

data_statement:
    RW_DATA data_list
;

data_list:
    data
    data_list optional_comma data
;

data:
    data_variable_list '/' data_constant_list '/'
;

data_variable_list:
    data_variable
    data_variable_list ',' data_variable
;

data_variable:
    variable
    {
        add_identifier_list( $1, REF, SET );
    }
    data IMPLIED DO_list
;

data IMPLIED DO_list:
    '(' data_variable_list ',' data_identifier '=' expression_list ')'
;

data_identifier:
    IDENTIFIER
    {
        add_identifier_list( $1, O, SET );
    }
;

data_constant_list:
    data_constant
    data_constant_list ',' data_constant
;

data_constant:
    data_initialization
    repetition_factor '*' data_initialization
;

repetition_factor:
    IDENTIFIER
    {
        add_identifier_list( $1, O, REF );
    }
    INTEGER
;

```

```

data_initialization:
  IDENTIFIER
  {
    add_identifier_list( $1, 0, REF );
  }
  |
  character_constant
  |
  logical_constant
  |
  signed_numerical_constant
  ;

```

```

signed_numerical_constant:
  numerical_constant
  |
  '+' numerical_constant %prec SIGN
  |
  '-' numerical_constant %prec SIGN
  ;

```

```

expression:
  parenthesis_expression
  {
    $$ = $1;
  }
  |
  simple_expression
  {
    $$ = $1;
  }
  ;

```

```

parenthesis_expression:
  '(' expression ')'
  {
    $$ = "";
  }
  ;

```

```

simple_expression:
  variable
  {
    $$ = $1;
  }
  |
  constant
  {
    $$ = $1;
  }
  |
  arithmetic_expression
  {
    $$ = $1;
  }
  |
  character_expression
  {
    $$ = $1;
  }
  |
  relational_expression
  {
    $$ = $1;
  }
  |
  logical_expression
  {
    $$ = $1;
  }
  |
  unary_expression
  {
    $$ = '1';
  }
  ;

```

```

variable:
    identifier
    {
        $$ = $1;
    }
    |
    identifier string_subset
    {
        $$ = $1;
    }
    |
    array
    {
        $$ = $1;
    }
    ;

array:
    identifier '(' optional_expression_list ')'
    {
        $$ = $1;
    }
    |
    identifier '(' optional_expression_list ')' string_subset
    {
        $$ = $1;
    }
    ;

identifier:
    IDENTIFIER
    {
        add_identifier_list( $1, 0, usage );
        if ( usage == SET ) usage = REF;
        $$ = $1;
    }
    ;

optional_expression_list:
    /* NULL */
    |
    expression_list
    ;

expression_list:
    expression
    |
    expression_list ',' expression
    ;

string_subset:
    '(' optional_expression ':' optional_expression ')'
    ;

optional_expression:
    /* NULL */
    |
    expression
    ;

constant:
    character_constant
    {
        $$ = $1;
    }
    |
    logical_constant
    {
        $$ = $1;
    }
    |
    numerical_constant

```

```

    {
        $$ = $1;
    }
;

logical_constant:
    RW_FALSE
    {
        $$ = "";
    }
|
    RW_TRUE
    {
        $$ = "";
    }
;

character_constant:
    HOLLERITH
    {
        $$ = "";
    }
|
    STRING
    {
        $$ = "";
    }
;

numerical_constant:
    DOUBLE_PRECISION
    {
        $$ = "";
    }
|
    INTEGER
    {
        $$ = "";
    }
|
    REAL
    {
        $$ = "";
    }
;

arithmetic_expression:
    expression '+' expression %prec '+'
    {
        $$ = "";
    }
|
    expression '-' expression %prec '-'
    {
        $$ = "";
    }
|
    expression '*' expression %prec '*'
    {
        $$ = "";
    }
;
    expression '/' expression %prec '/'
    {
        $$ = "";
    }
|
    expression EXPONENTIATE expression %prec EXPONENTIATE
    {
        $$ = "";
    }
;

character_expression:
    expression '/' '/' expression %prec CONCATENATE
    {

```

```

    $$ = "";
}
;

relational_expression:
    expression RW_EQ expression %prec RW_EQ
    {
        $$ = "";
    }
    |
    expression RW_NE expression %prec RW_NE
    {
        $$ = "";
    }
    |
    expression RW_LT expression %prec RW_LT
    {
        $$ = "";
    }
    |
    expression RW_LE expression %prec RW_LE
    {
        $$ = "";
    }
    |
    expression RW_GT expression %prec RW_GT
    {
        $$ = "";
    }
    |
    expression RW_GE expression %prec RW_GE
    {
        $$ = "";
    }
;

logical_expression:
    expression RW_AND expression %prec RW_AND
    {
        $$ = "";
    }
    |
    expression RW_OR expression %prec RW_OR
    {
        $$ = "";
    }
    |
    expression RW_EQV expression %prec RW_EQV
    {
        $$ = "";
    }
    |
    expression RW_NEQV expression %prec RW_NEQV
    {
        $$ = "";
    }
;

unary_expression:
    '+' expression %prec SIGN
    {
        $$ = "";
    }
    |
    '-' expression %prec SIGN
    {
        $$ = "";
    }
    |
    RW_NOT expression %prec RW_NOT
    {
        $$ = "";
    }
;

executable_statement:
    do_statement

```

```

|
|   logical_if_statement
|
|   block_if_statement
|
|   else_statement
|
|   else_if_statement
|
|   end_if_statement
|
|   subset_executable_statement
;

do_statement:
  RW_DO INTEGER do_identifier '=' expression_list
;

do_identifier:
  IDENTIFIER
  {
    add_identifier_list( $1, 0, SET );
  }
;

logical_if_statement:
  if_expression subset_executable_statement
  {
    level--;
  }
;

if_expression:
  RW_IF '(' expression ')'
  {
    level++;
  }
;

block_if_statement:
  RW_IF '(' expression ')' RW_THEN
  {
    level++;
  }
;

else_statement:
  RW_ELSE
  {
    level--;
    level++;
  }
;

else_if_statement:
  RW_ELSE_IF '(' expression ')' RW_THEN
  {
    level--;
    level++;
  }
;

end_if_statement:
  RW_END_IF
  {
    level--;
  }
;

subset_executable_statement:
  assignment_statement
|

```

```

    assign_statement
  |
    arithmetic_if_statement
  |
    continue_statement
  |
    call_statement
  |
    return_statement
  |
    unconditional_go_to_statement
  |
    computed_go_to_statement
  |
    assigned_go_to_statement
  |
    stop_statement
  |
    pause_statement
  |
    io_statement
  ;

assignment_statement:
    { if ( usage == REF ) usage = SET; } variable '=' expression
  ;

assign_statement:
    RW_ASSIGN INTEGER RW_TO IDENTIFIER
  ;

arithmetic_if_statement:
    RW_IF '(' expression ')' integer_list
  ;

continue_statement:
    RW_CONTINUE
  ;

call_statement:
    RW_CALL call_identifier
  |
    RW_CALL call_identifier optional_actual_argument_list
  ;

call_identifier:
    IDENTIFIER
    {
        add_statement_list( $1, 0 );
    }
  ;

optional_actual_argument_list:
    '(' '(' ')'
  |
    '('
        { usage &= ~REF; }
        actual_argument_list
        { usage |= REF; }
    ')'
  ;

actual_argument_list:
    actual_argument
  |
    actual_argument_list ',' actual_argument
  ;

actual_argument:
    { argument_number++; } expression
  |
    { argument_number++; } actual_argument_alternate_return

```

```

;

actual_argument_alterate_return:
    '*' INTEGER
    {
        $$ = "";
    }
;

return_statement:
    RW_RETURN optional_expression
;

unconditional_go_to_statement:
    RW_GO_TO INTEGER
;

computed_go_to_statement:
    RW_GO_TO '(' integer_list ')' optional_comma expression
;

assigned_go_to_statement:
    RW_GO_TO IDENTIFIER
    |
    RW_GO_TO IDENTIFIER optional_comma '(' integer_list ')'
;

integer_list:
    INTEGER
    |
    integer_list ',' INTEGER
;

optional_comma:
    /* NULL */
    |
    ','
;

pause_statement:
    RW_PAUSE optional_expression
;

stop_statement:
    RW_STOP optional_expression
;

io_statement:
    open_statement
    |
    close_statement
    |
    inquire_statement
    |
    read_statement
    |
    write_statement
    |
    print_statement
    |
    backspace_statement
    |
    rewind_statement
    |
    endfile_statement
;

open_statement:
    RW_OPEN '(' control_information_list ')'
;

```



```

close_statement:
    RW_CLOSE '(' control_information_list ')'
    ;

inquire_statement:
    RW_INQUIRE '(' control_information_list ')'
    ;

read_statement:
    RW_READ '(' control_information_list ')' optional_io_list
    |
    RW_READ control
    |
    RW_READ control ',' io_list
    ;

write_statement:
    RW_WRITE '(' control_information_list ')' optional_io_list
    ;

print_statement:
    RW_PRINT control
    |
    RW_PRINT control ',' io_list
    ;

backspace_statement:
    RW_BACKSPACE '(' control_information_list ')'
    |
    RW_BACKSPACE control
    ;

rewind_statement:
    RW_REWIND '(' control_information_list ')'
    |
    RW_REWIND control
    ;

endfile_statement:
    RW_ENDFILE '(' control_information_list ')'
    |
    RW_ENDFILE control
    ;

control_information_list:
    control_information
    |
    control_information_list ',' control_information
    ;

control_information:
    control
    |
    IDENTIFIER '=' expression
    ;

control:
    variable
    |
    constant
    |
    '*'
    ;

optional_io_list:
    /* NULL */
    |
    io_list

```

```

;

io_list:
    io
    |
    io_list ',' io
;

io:
    expression
    |
    io_implied_do_list
;

io_implied_do_list:
    '(' io_list ',' io_identifier '=' expression_list ')'

io_identifier:
    IDENTIFIER
    {
        add_identifier_list( $1, 0, SET );
    }
;

format_statement:
    RW_FORMAT
;

%%

FILE: initial/include/list.h

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#define LIST struct list
LIST
{
    char *identifier;
    int line;
    int usage;
    int argument_number;
    LIST *argument_list;
    LIST *next;
};

extern LIST *end_list( );
extern LIST *add_list_forward( );
extern LIST *add_list_reverse( );
extern LIST *find_list( );
extern void add_formal_argument_list( );
extern void add_statement_list( );
extern LIST *add_list( );
extern void add_identifier_list( );
extern void begin_block( );
extern void end_block( );
extern void usage_actual_argument_list( );
extern void usage_formal_argument_list( );
extern int find_block_number( );
extern void usage_block( );

#define MAXIMUM_NUMBER_BLOCK 1024
extern char *block[ MAXIMUM_NUMBER_BLOCK ];
extern int recursive[ MAXIMUM_NUMBER_BLOCK ];
extern LIST *formal_argument_list[ MAXIMUM_NUMBER_BLOCK ];
extern LIST *variable_list[ MAXIMUM_NUMBER_BLOCK ];

```

```
extern LIST *statement_list[ MAXIMUM_NUMBER_BLOCK ];
extern int number_block;
```

```
FILE: initial/include/usage.h
```

```
/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */
```

```
#define REF 0x1
#define SET 0x2
#define CONDITIONAL 0x4
#define INITIALIZED 0x8
```

```
FILE: initial/library/Makefile
```

```
#
# Copyright 1991
# Georgia Institute of Technology
# Computer Engineering Research Laboratory
# Author: Stephen R. Wachtel
#
```

```
CC = cc -g
INCLUDE = ../include
CFLAGS = -I$(INCLUDE)
LIBRARY = library.a
```

```
OBJECTS = \
    duplicate.o \
    hollerith.o \
    intrinsic.o \
    link_list.o \
    main.o \
    non_blank.o \
    summary.o \
    uppercase.o \
    yyerror.o \
    yygetc.o \
    yywrap.o
```

```
$(LIBRARY): $(OBJECTS)
    ar crv $(LIBRARY) $(OBJECTS)
    ranlib $(LIBRARY)
```

```
.SUFFIXES: .c .o
.c.o:
    $(CC) -c $(CFLAGS) $<
```

```
clean:
    rm -f $(LIBRARY) $(OBJECTS)
```

```
FILE: initial/library/duplicate.c
```

```
/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */
```

```
#include <stdio.h>
#include <string.h>
#include <malloc.h>
```

```

char *duplicate( string )
register char *string;
{
    register char *temporary = (char *)NULL;

    if ( string != (char *)NULL )
    {
        if ( ( temporary = (char *)malloc( strlen( string ) + 1 ) ) != (char *)NULL )
            strcpy( temporary, string );
        else
            fprintf( stderr, "ERROR: duplicate( %s )\n", string );
    }

    return( temporary );
} /* duplicate */

```

FILE: initial/library/hollerith.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```
#include <stdio.h>
```

```

char *hollerith( string, delimiter )
register char *string;
register char delimiter;
{
    int hollerith_length;
    register int string_length = 0;

    sscanf( string, "%dh", &hollerith_length );

    string[ string_length++ ] = delimiter;
    while ( hollerith_length != 0 )
    {
        if ( ( string[ string_length ] = yyinput( ) ) == '\n' )
        {
            yyunput( string[ string_length ] );
            break;
        }

        string_length++;
        hollerith_length--;
    }
    string[ string_length++ ] = delimiter;

    string[ string_length ] = '\0';
    return( string );
} /* hollerith */

```

FILE: initial/library/intrinsic.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```

extern char *duplicate( );
extern char *uppercase( );

```

```

static char *intrinsic_table[ ] =
{
    "ABS",
    "ACOS",
    "AIMAG",
    "AINT",

```

"ALOG",  
"ALOG10",  
"AMAX0",  
"AMAX1",  
"AMIN0",  
"AMIN1",  
"AMOD",  
"ANINT",  
"ASIN",  
"ATAN",  
"ATAN2",  
"CABS",  
"CCOS",  
"CEXP",  
"CHAR",  
"CLOG",  
"CMLPX",  
"CONJG",  
"COS",  
"COSH",  
"CSIN",  
"CSQRT",  
"DABS",  
"DACOS",  
"DASIN",  
"DATAN",  
"DATAN2",  
"DBLE",  
"DCOS",  
"DCOSH",  
"DDIM",  
"DEXP",  
"DIM",  
"DINT",  
"DLOG",  
"DLOG10",  
"DMAX1",  
"DMIN1",  
"DMOD",  
"DNINT",  
"DPROD",  
"DSIGN",  
"DSIN",  
"DSINH",  
"DSQRT",  
"DTAN",  
"DTANH",  
"EXP",  
"FLOAT",  
"IABS",  
"ICHAR",  
"IDIM",  
"IDINT",  
"IDNINT",  
"IFIX",  
"INDEX",  
"INT",  
"ISIGN",  
"LEN",  
"LGE",  
"LGT",  
"LLE",  
"LLT",  
"LOG",  
"LOG10",  
"MAX",  
"MAX0",  
"MAX1",  
"MIN",  
"MIN0",  
"MIN1",  
"MOD",  
"NINT",  
"REAL",  
"SIGN",  
"SIN",  
"SINH",  
"SNGL",  
"SQRT",  
"TAN",  
"TANH"

```

};

#define INTRINSIC_TABLE ( sizeof( intrinsic_table ) / sizeof( char * ) )

int intrinsic( identifier )
register char *identifier;
{
    register int low, high;
    register int middle, test;
    register char *temporary = duplicate( identifier );

    low = 0;
    high = INTRINSIC_TABLE - 1;

    uppercase( temporary );

    while ( low <= high )
    {
        middle = ( low + high ) / 2;
        test = strcmp( temporary, intrinsic_table[ middle ] );

        if ( test < 0 )
        {
            high = middle - 1;
            continue;
        }

        if ( test > 0 )
        {
            low = middle + 1;
            continue;
        }

        free( temporary );
        return( 1 );
    }

    free( temporary );
    return( 0 );
} /* intrinsic */

```

FILE: initial/library/link\_list.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include <stdio.h>
#include <malloc.h>
#include <string.h>
#include "list.h"
#include "usage.h"

extern int yylineno;
extern int argument_number;

char *block[ MAXIMUM_NUMBER_BLOCK ];
int recursive[ MAXIMUM_NUMBER_BLOCK ];
LIST *formal_argument_list[ MAXIMUM_NUMBER_BLOCK ];
LIST *variable_list[ MAXIMUM_NUMBER_BLOCK ];
LIST *statement_list[ MAXIMUM_NUMBER_BLOCK ];
int number_block = 0;

#define FORWARD 0
#define REVERSE 1
static int order = FORWARD;

LIST *end_list( list )
register LIST *list;

```

```

{
    if ( list != (LIST *)NULL )
    {
        while ( list->next != (LIST *)NULL )
            list = list->next;
    }

    return( list );
} /* end_list */

LIST *add_list_forward( list, identifier )
register LIST **list;
register char *identifier;
{
    register LIST *temporary = (LIST *)malloc( sizeof( LIST ) );

    temporary->identifier = identifier;
    temporary->line = yylineno;
    temporary->usage = 0;
    temporary->argument_number = argument_number;
    temporary->argument_list = (LIST *)NULL;
    temporary->next = (LIST *)NULL;

    if ( *list == (LIST *)NULL )
        *list = temporary;
    else
        end_list( *list )->next = temporary;

    return( temporary );
} /* add_list_forward */

LIST *add_list_reverse( list, identifier )
register LIST **list;
register char *identifier;
{
    register LIST *temporary = (LIST *)malloc( sizeof( LIST ) );

    temporary->identifier = identifier;
    temporary->line = yylineno;
    temporary->usage = 0;
    temporary->argument_number = argument_number;
    temporary->argument_list = (LIST *)NULL;
    temporary->next = *list;

    *list = temporary;

    return( temporary );
} /* add_list_reverse */

LIST *find_list( list, identifier )
register LIST *list;
register char *identifier;
{
    register LIST *temporary = (LIST *)NULL;

    while ( list != (LIST *)NULL )
    {
        if ( strcmp( list->identifier, identifier ) == 0 )
            temporary = list;

        list = list->next;
    }

    return( temporary );
} /* find_list */

void add_formal_argument_list( identifier )
register char *identifier;
{
    argument_number++;
    add_list_forward( &formal_argument_list[ number_block ], identifier );
} /* add_formal_argument_list */

void add_statement_list( identifier, usage )
register char *identifier;
register int usage;

```

```

{
    argument_number = 0;
    if ( identifier != (char *)NULL )
    {
        end_list( statement_list[ number_block ] )->identifier = identifier;

        order = FORWARD;
    }
    else
    {
        add_list_forward( &statement_list[ number_block ], identifier )->usage = usage;

        order = REVERSE;
    }
} /* add_statement_list */

LIST *add_list( identifier, block_number )
register char *identifier;
register int block_number;
{
    register LIST *temporary;

    temporary = find_list( formal_argument_list[ block_number ], identifier );
    if ( temporary != (LIST *)NULL )
        return( temporary );

    temporary = find_list( variable_list[ block_number ], identifier );
    if ( temporary != (LIST *)NULL )
        return( temporary );

    temporary = add_list_forward( &variable_list[ block_number ], identifier );
    return( temporary );
} /* add_list */

void add_identifier_list( identifier, and_usage, or_usage )
register char *identifier;
unsigned int and_usage;
unsigned int or_usage;
{
    register LIST *temporary;

    switch ( order )
    {
        case FORWARD:
            if ( and_usage != 0 )
            {
                temporary = find_list( end_list( statement_list[ number_block ] )->
>argument_list, identifier );
                temporary->usage &= ~and_usage;
            }
            else
                temporary = add_list_forward( &end_list( statement_list[ number_block ] )->
>argument_list, identifier );
            break;

        case REVERSE:
            if ( and_usage != 0 )
            {
                temporary = find_list( end_list( statement_list[ number_block ] )->
>argument_list, identifier );
                temporary->usage &= ~and_usage;
            }
            else
                temporary = add_list_reverse( &end_list( statement_list[ number_block ] )->
>argument_list, identifier );
            break;
    }

    temporary->usage |= or_usage;
} /* add_identifier_list */

void begin_block( identifier )
register char *identifier;
{
    block[ number_block ] = identifier;
    recursive[ number_block ] = 0;
    formal_argument_list[ number_block ] = (LIST *)NULL;
}

```



```

    variable_list[ number_block ] = (LIST *)NULL;
    statement_list[ number_block ] = (LIST *)NULL;
    argument_number = 0;
} /* begin_block */

void end_block( )
{
    number_block++;
} /* end_block */

void usage_actual_argument_list( block_number, actual_argument_list, usage )
register int block_number;
register LIST *actual_argument_list;
register int usage;
{
    register LIST *temporary;

    while ( actual_argument_list != (LIST *)NULL )
    {
        temporary = add_list( actual_argument_list->identifier, block_number );
        temporary->usage |= usage;

        switch ( temporary->usage )
        {
            case ( 0 ):
            case ( CONDITIONAL ):
                temporary->usage |= actual_argument_list->usage;
                if ( ( usage & CONDITIONAL ) == CONDITIONAL )
                    temporary->usage &= ~INITIALIZED;

                if ( ( temporary->usage & CONDITIONAL ) != CONDITIONAL )
                    if ( ( ( temporary->usage & SET ) == SET ) && ( ( temporary->usage &
REF ) != REF ) )
                        temporary->usage |= INITIALIZED;
                break;

            case ( REF ):
            case ( REF | INITIALIZED ):
                if ( ( actual_argument_list->usage & CONDITIONAL ) == CONDITIONAL )
                    temporary->usage |= CONDITIONAL;

            case ( REF | CONDITIONAL ):
            case ( REF | CONDITIONAL | INITIALIZED ):
                if ( ( actual_argument_list->usage & SET ) == SET )
                    temporary->usage |= SET;
                break;

            case ( SET ):
            case ( SET | INITIALIZED ):
                if ( ( actual_argument_list->usage & CONDITIONAL ) == CONDITIONAL )
                    temporary->usage |= CONDITIONAL;

            case ( SET | CONDITIONAL ):
            case ( SET | INITIALIZED | CONDITIONAL ):
                if ( ( actual_argument_list->usage & REF ) == REF )
                    temporary->usage |= REF;
                break;

            case ( REF | SET ):
            case ( REF | SET | CONDITIONAL ):

            case ( SET | REF | INITIALIZED ):
            case ( SET | REF | CONDITIONAL | INITIALIZED ):
                break;
        }

        actual_argument_list = actual_argument_list->next;
    }
} /* usage_actual_argument_list */

void usage_formal_argument_list( actual_argument_list, formal_argument_list )
register LIST *actual_argument_list;
register LIST *formal_argument_list;
{
    while ( ( actual_argument_list != (LIST *)NULL ) && ( formal_argument_list != (LIST

```

```

*)NULL ) )
{
    if ( actual_argument_list->argument_number == formal_argument_list-
>argument_number )
    {
        actual_argument_list->usage = formal_argument_list->usage;
        actual_argument_list = actual_argument_list->next;
    }
    else
        formal_argument_list = formal_argument_list->next;
}
} /* use_arg_formal_argument_list */

int find_block_number( identifier )
register char *identifier;
{
    register int block_number;

    for ( block_number = 0; block_number < number_block; block_number++ )
    {
        if ( strcmp( block[ block_number ], identifier ) == 0 )
            return( block_number );
    }

    fprintf( stderr, "WARNING: block %s not found\n", identifier );
    begin_block( identifier );
    end_block( );

    return( number_block - 1 );
} /* find_block_number */

void usage_block( block_number, actual_argument_list )
register int block_number;
register LIST *actual_argument_list;
{
    register LIST *statement = statement_list[ block_number ];

    if ( recursive[ block_number ] != 0 )
    {
        fprintf( stderr, "ERROR: block %s recursive\n", block[ block_number ] );
        exit( -1 );
    }
    recursive[ block_number ] = 1;

    while ( statement != (LIST *)NULL )
    {
        if ( statement->identifier != (char *)NULL )
            usage_block( find_block_number( statement->identifier ), statement-
>argument_list );

        usage_actual_argument_list( block_number, statement->argument_list, statement-
>usage );
        statement = statement->next;
    }

    usage_formal_argument_list( actual_argument_list, formal_argument_list[ block_number ]
);
    recursive[ block_number ] = 0;
} /* usage_block */

FILE: initial/library/main.c

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include <stdio.h>
#include <string.h>
#include "usage.h"

extern FILE *yyin;
extern FILE *yyout;

```

```
extern int conditional;
```

```
#define PROGRAM argument[ 0 ]
#define INPUT_FILE argument[ 1 ]
#define OUTPUT_FILE argument[ 2 ]
```

```
int main( number_argument, argument )
int number_argument;
char *argument[ ];
{
loop:
    if ( strcmp( argument[ number_argument - 1 ], "-conditional=y" ) == 0 )
    {
        number_argument--;
        conditional &= ~CONDITIONAL;
        goto loop;
    }

    if ( strcmp( argument[ number_argument - 1 ], "-conditional=n" ) == 0 )
    {
        number_argument--;
        conditional |= CONDITIONAL;
        goto loop;
    }

    if ( number_argument == 1 )
    {
        yyin = stdin;
        yyout = stdout;

        yyparse( );
        exit( 0 );
    }

    if ( number_argument == 3 )
    {
        if ( ( yyin = fopen( INPUT_FILE, "r" ) ) == (FILE *)NULL )
        {
            fprintf( stderr, "%s: ERROR - unable to open input file '%s'\n", PROGRAM,
INPUT_FILE );
            exit( -1 );
        }

        if ( ( yyout = fopen( OUTPUT_FILE, "w" ) ) == (FILE *)NULL )
        {
            fprintf( stderr, "%s: ERROR - unable to open output file '%s'\n", PROGRAM,
OUTPUT_FILE );
            exit( -1 );
        }

        yyparse( );
        exit( 0 );
    }

    fprintf( stderr, "usage: %s <input file> <output file> [-conditional=y or n]\n",
PROGRAM );
    exit( 0 );
} /* main */
```

```
FILE: initial/library/non_blank.c
```

```
/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */
```

```
#include <string.h>
```

```
char *non_blank( string )
register char *string;
{
    register int offset;
    register int length;
```

```

length = strlen( string ) - 1;
while ( ( string[ length ] == ' ' ) && ( string[ length ] != '\0' ) )
    string[ length-- ] = '\0';

offset = 0;
while ( ( string[ offset ] == ' ' ) && ( string[ offset ] != '\0' ) )
    string[ offset++ ] = '\0';

strcpy( string, &string[ offset ] );

if ( strlen( string ) != 0 )
    return( string );
else
    return( 0 );
} /* non_blank */

```

FILE: initial/library/summary.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include <stdio.h>
#include <string.h>
#include "list.h"
#include "usage.h"

extern FILE *yyin;
extern FILE *yyout;

void print_list( file, string, list )
register FILE *file;
register char *string;
register LIST *list;
{
    int column = strlen( string );
    char buffer[ 256 ];

    fprintf( file, "%s", string );

    while ( list != (LIST *)NULL )
    {
        fprintf( file, "%s", list->identifier );
        column += strlen( list->identifier );

        if ( list->usage != 0 )
        {
            fprintf( file, "(" );
            column++;

            if ( ( list->usage & CONDITIONAL ) == CONDITIONAL )
            {
                fprintf( file, "C" );
                column++;
            }

            if ( ( list->usage & INITIALIZED ) == INITIALIZED )
            {
                if ( ( list->usage & SET ) == SET )
                {
                    fprintf( file, "S" );
                    column++;
                }

                if ( ( list->usage & REF ) == REF )
                {
                    fprintf( file, "R" );
                    column++;
                }
            }
        }
        else
        {

```

```

        if ( ( list->usage & REF ) == REF )
        {
            fprintf( file, "R" );
            column++;
        }

        if ( ( list->usage & SET ) == SET )
        {
            fprintf( file, "S" );
            column++;
        }
    }

    fprintf( file, ")" );
    column++;
}

if ( list->argument_number != 0 )
{
    sprintf( buffer, "%d", list->argument_number );
    fprintf( file, "%s", buffer );
    column += strlen( buffer );
}

if ( list->next != (LIST *)NULL )
{
    fprintf( file, "," );
    column++;
}

#define MAXIMUM_COLUMN 72
    if ( column >= MAXIMUM_COLUMN )
    {
        fprintf( file, "\n" );
        fprintf( file, "\t\t" );
        column = 8;
    }

    list = list->next;
}
/* print_list */

void print_statement_list( file, list )
register FILE *file;
register LIST *list;
{
    register char string[ 256 ];

    while ( list != (LIST *)NULL )
    {
        if ( list->identifier != (char *)NULL )
        {
            if ( ( list->usage & CONDITIONAL ) == CONDITIONAL )
                sprintf( string, "    line %d{C}, %s(", list->line, list->identifier );
            else
                sprintf( string, "    line %d, %s(", list->line, list->identifier );
            print_list( file, string, list->argument_list );
            fprintf( file, ")\n" );
        }
        else
        {
            if ( list->argument_list != (LIST *)NULL )
            {
                if ( ( list->usage & CONDITIONAL ) == CONDITIONAL )
                    sprintf( string, "    line %d{C}, ", list->line );
                else
                    sprintf( string, "    line %d, ", list->line );
                print_list( file, string, list->argument_list );
                fprintf( file, "\n" );
            }
        }

        list = list->next;
    }
}
/* print_statement_list */

void print_variable_list( file, block_number )
register FILE *file;
register int block_number;

```

```

{
    register LIST *variable = variable_list[ block_number ];
    while ( variable != (LIST *)NULL )
    {
#ifdef DEBUG
        fprintf( file, "%s ", variable->identifier );

        if ( ( variable->usage & CONDITIONAL ) == CONDITIONAL )
            fprintf( file, "C" );
        else
            fprintf( file, "-" );

        if ( ( variable->usage & INITIALIZED ) == INITIALIZED )
        {
            if ( ( variable->usage & SET ) == SET )
                fprintf( file, "S" );
            else
                fprintf( file, "-" );

            if ( ( variable->usage & REF ) == REF )
                fprintf( file, "R" );
            else
                fprintf( file, "-" );
        }
        else
        {
            if ( ( variable->usage & REF ) == REF )
                fprintf( file, "R" );
            else
                fprintf( file, "-" );

            if ( ( variable->usage & SET ) == SET )
                fprintf( file, "S" );
            else
                fprintf( file, "-" );
        }

        fprintf( file, "\n" );
    #else
        if ( ( variable->usage & INITIALIZED ) != INITIALIZED )
        {
            if ( ( variable->usage & REF ) == REF )
            {
                if ( ( variable->usage & SET ) == SET )
                    fprintf( file, "%s RS\n", variable->identifier );
                else
                    fprintf( file, "%s R-\n", variable->identifier );
            }
        }
    #endif

        variable = variable->next;
    }
} /* print_variable_list */

void summary( )
{
    register int block_number;
    char string[ 256 ];

    usage_block( 0, 0 );

    for ( block_number = 0; block_number != number_block; block_number++ )
    {
        sprintf( string, "%s ", block[ block_number ] );
        print_list( yyout, string, formal_argument_list[ block_number ] );
        fprintf( yyout, "\n" );

        print_variable_list( yyout, block_number );
        fprintf( yyout, "\n" );

#ifdef DEBUG
        print_statement_list( yyout, statement_list[ block_number ] );
        fprintf( yyout, "\n" );
    #endif
    }
} /* summary */

```

FILE: initial/library/uppercase.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

char *uppercase( string )
register char *string;
{
    register int index = 0;

    while ( string[ index ] != '\0' )
    {
        string[ index ] = toupper( string[ index ] );
        index++;
    }

    return( string );
} /* uppercase */

```

FILE: initial/library/yyerror.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include <stdio.h>

extern int yylineno;

void yyerror( string )
register char *string;
{
    fprintf( stderr, "line %d, %s\n", yylineno, string );

    exit( -1 );
} /* yyerror */

```

FILE: initial/library/yygetc.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include <stdio.h>
#include <ctype.h>

extern int yylineno;

int tab( length )
register int length;
{
    while ( length-- != 0 )
        yyunput( ' ' );

    return( ' ' );
} /* tab */

int yygetc( file )

```

```

register FILE *file;
{
    int c;
    int column[ 6 ];

loop:
    if ( ( c = getc( file ) ) == '\t' )
        c = tab( 6 );

    if ( c != '\n' )
        return( c );

    if ( ( column[ 0 ] = getc( file ) ) != ' ' )
        goto abort_0;
    if ( ( column[ 1 ] = getc( file ) ) != ' ' )
        goto abort_1;
    if ( ( column[ 2 ] = getc( file ) ) != ' ' )
        goto abort_2;
    if ( ( column[ 3 ] = getc( file ) ) != ' ' )
        goto abort_3;
    if ( ( column[ 4 ] = getc( file ) ) != ' ' )
        goto abort_4;
    if ( isspace( column[ 5 ] = getc( file ) ) )
        goto abort_5;

    yylineno++;
    goto loop;

abort_5:
    if ( column[ 5 ] == '\t' )
        tab( 1 );
    else
    {
        yyunput( column[ 5 ] );
        if ( column[ 5 ] == '\n' )
            yylineno++;
    }

abort_4:
    if ( column[ 4 ] == '\t' )
        tab( 2 );
    else
    {
        yyunput( column[ 4 ] );
        if ( column[ 4 ] == '\n' )
            yylineno++;
    }

abort_3:
    if ( column[ 3 ] == '\t' )
        tab( 3 );
    else
    {
        yyunput( column[ 3 ] );
        if ( column[ 3 ] == '\n' )
            yylineno++;
    }

abort_2:
    if ( column[ 2 ] == '\t' )
        tab( 4 );
    else
    {
        yyunput( column[ 2 ] );
        if ( column[ 2 ] == '\n' )
            yylineno++;
    }

abort_1:
    if ( column[ 1 ] == '\t' )
        tab( 5 );
    else
    {
        yyunput( column[ 1 ] );
        if ( column[ 1 ] == '\n' )
            yylineno++;
    }

abort_0:
    if ( column[ 0 ] == '\t' )
        tab( 6 );

```



```

    else
    {
        yyunput( column[ 0 ] );
        if ( column[ 0 ] == '\n' )
            yylineno++;
    }

    return( c );
} /* yygetc */

```

FILE: initial/library/yywrap.

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```

int yywrap( )
{
    return( 1 );
} /* yywrap */

```

FILE: initial/scanner.l

```

%{
/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */
%}

```

```

%a 10000
%e 10000
%k 10000
%n 10000
%o 10000
%p 10000

```

```

a  [aA]
b  [bB]
c  [cC]
d  [dD]
e  [eE]
f  [fF]
g  [gG]
h  [hH]
i  [iI]
j  [jJ]
k  [kK]
l  [lL]
m  [mM]
n  [nN]
o  [oO]
p  [pP]
q  [qQ]
r  [rR]
s  [sS]
t  [tT]
u  [uU]
v  [vV]
w  [wW]
x  [xX]
y  [yY]
z  [zZ]

```

```

%{
#include "grammar.h"
extern char *yylval;

```

```

#undef YYLMAX
#define YYLMAX (256*20)

extern char *duplicate( );
extern char *hollerith( );
extern char *non_blank( );
extern char *uppercase( );
%}

%%

^[\\*cC].*[\\n] |
^([\\ ])*[\\n] {
#ifdef DEBUG
    ECHO;
#endif
    yylval = duplicate( yytext );
    return( COMMENT );
}

[\\ ] {
#ifdef DEBUG
    ECHO;
#endif
    /* return( '\\ ' ) */;
}

[\\&] {
#ifdef DEBUG
    ECHO;
#endif
    return( '\\&' );
}

[\\(] {
#ifdef DEBUG
    ECHO;
#endif
    return( '\\(' );
}

[\\)] {
#ifdef DEBUG
    ECHO;
#endif
    return( '\\)' );
}

[\\*] {
#ifdef DEBUG
    ECHO;
#endif
    return( '\\*' );
}

[\\*]\\{\\*\\* {
#ifdef DEBUG
    ECHO;
#endif
    return( EXPONENTIATE );
}

[\\+] {
#ifdef DEBUG
    ECHO;
#endif
    return( '\\+' );
}

```

```
[\.] {
#ifdef DEBUG
    ECHO;
#endif
    return( '\,' );
}
```

```
[\-] {
#ifdef DEBUG
    ECHO;
#endif
    return( '\-' );
}
```

```
[\.] {
#ifdef DEBUG
    ECHO;
#endif
    return( '\.' );
}
```

```
[/] {
#ifdef DEBUG
    ECHO;
#endif
    return( '/' );
}
```

```
[\:] {
#ifdef DEBUG
    ECHO;
#endif
    return( ':' );
}
```

```
[\=] {
#ifdef DEBUG
    ECHO;
#endif
    return( '=' );
}
```

```
[\n] {
#ifdef DEBUG
    ECHO;
#endif
    /* return( '\n' ) */;
}
```

```
[\t] {
#ifdef DEBUG
    ECHO;
#endif
    /* return( '\t' ) */;
}
```

```
[\.]{a}{n}{d}[\.] {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_AND );
}
```

```
[\.]{e}{q}[\.] {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_EQ );
}
```

```
[\.]{e}{q}{v}[\.] {
```

```

#ifdef DEBUG
    ECHO;
#endif
    return( RW_EQV );
}

[\.]{f}{a}{l}{s}{e}[\.] {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_FALSE );
}

[\.]{g}{e}[\.] {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_GE );
}

[\.]{g}{t}[\.] {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_GT );
}

[\.]{l}{e}[\.] {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_LE );
}

[\.]{l}{t}[\.] {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_LT );
}

[\.]{n}{e}[\.] {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_NE );
}

[\.]{n}{e}{q}{v}[\.] {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_NEQV );
}

[\.]{n}{o}{t}[\.] {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_NOT );
}

[\.]{o}{r}[\.] {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_OR );
}

[\.]{t}{r}{u}{e}[\.] {
#ifdef DEBUG

```

```

    ECHO;
#endif
    return( RW_TRUE );
}

{a}{s}{s}{i}{g}{n} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_ASSIGN );
}

{b){a){c){k){s){p){a){c){e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_BACKSPACE );
}

{b){l){o){c){k){\ }*(d){a){t){a} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_BLOCK_DATA );
}

{c){a){l){l} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_CALL );
}

{c){h){a){r){a){c){t){e){r} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_CHARACTER );
}

{c){l){o){s){e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_CLOSE );
}

{c){o){m){m){o){n} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_COMMON );
}

{c){o){m){p){l){e){x} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_COMPLEX );
}

{c){o){n){t){i){n){u){e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_CONTINUE );
}

{d){a){t){a} {
#ifdef DEBUG
    ECHO;

```

```

#endif
    return( RW_DATA );
}

{d}{i}{m}{e}{n}{s}{i}{o}{n} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_DIMENSION );
}

{d}{o} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_DO );
}

{d}{o}{u}{b}{l}{e}{\ }*{p}{r}{e}{c}{i}{s}{i}{o}{n} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_DOUBLE_PRECISION );
}

{e}{l}{s}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_ELSE );
}

{e}{l}{s}{e}{\ }*{i}{f} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_ELSE_IF );
}

{e}{n}{d} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_END );
}

{e}{n}{d}{\ }*{i}{f} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_END_IF );
}

{e}{n}{d}{f}{i}{l}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_ENDFILE );
}

{e}{n}{t}{r}{y} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_ENTRY );
}

{e}{q}{u}{i}{v}{a}{l}{e}{n}{c}{e} {
#ifdef DEBUG
    ECHO;
#endif
}

```

```

    return( RW_EQUIVALENCE );
}

{e}{x}{t}{e}{r}{n}{a}{l} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_EXTERNAL );
}

{f}{o}{r}{m}{a}{t}.* {
#ifdef DEBUG
    ECHO;
#endif
    yyval = duplicate( yytext );
    return( RW_FORMAT );
}

{f}{u}{n}{c}{t}{i}{o}{n} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_FUNCTION );
}

{g}{o}{[ \ ]*{t}{o} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_GO_TO );
}

{i}{f} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_IF );
}

{i}{m}{p}{l}{i}{c}{i}{t} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_IMPLICIT );
}

{i}{n}{c}{l}{u}{d}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_INCLUDE );
}

{i}{n}{q}{u}{i}{r}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_INQUIRE );
}

{i}{n}{t}{e}{g}{e}{r} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_INTEGER );
}

{i}{n}{t}{r}{i}{n}{s}{i}{c} {
#ifdef DEBUG
    ECHO;
#endif

```

```

    return( RW_INTRINSIC );
}

{l}{o}{g}{i}{c}{a}{l} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_LOGICAL );
}

{n}{a}{m}{e}{l}{i}{s}{t} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_NAMELIST );
}

{o}{p}{e}{n} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_OPEN );
}

{p}{a}{r}{a}{m}{e}{t}{e}{r} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_PARAMETER );
}

{p}{a}{u}{s}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_PAUSE );
}

{p}{r}{i}{n}{t} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_PRINT );
}

{p}{r}{o}{g}{r}{a}{m} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_PROGRAM );
}

{r}{e}{a}{d} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_READ );
}

{r}{e}{a}{l} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_REAL );
}

{r}{e}{t}{u}{r}{n} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_RETURN );
}

```



```

    }

{r}{e}{w}{i}{n}{d} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_REWIND );
}

{s}{a}{v}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_SAVE );
}

{s}{t}{o}{p} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_STOP );
}

{s}{u}{b}{r}{o}{u}{t}{i}{n}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_SUBROUTINE );
}

{t}{h}{e}{n} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_THEN );
}

{t}{o} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_TO );
}

{w}{r}{i}{t}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_WRITE );
}

{u}{n}{d}{e}{f}{i}{n}{e}{d} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_UNDEFINED );
}

[%a-zA-Z][_a-zA-Z0-9]* {
#ifdef DEBUG
    ECHO;
#endif
    yyval = duplicate( uppercase( yytext ) );
    return( IDENTIFIER );
}

^[0-9][0-9][0-9][0-9][0-9][\ ] {
#ifdef DEBUG
    ECHO;
#endif
    yyval = duplicate( non_blank( yytext ) );

```

```

    return( LABEL );
}

[0-9]+ |
[0-9]+/\.[a-zA-Z]+\.\. {
#ifdef DEBUG
    ECHO;
#endif
    yylval = duplicate( yytext );
    return( INTEGER );
}

[0-9]+\.[0-9]*([eE][\+\-]?[0-9]+)? |
[0-9]*\.[0-9]+([eE][\+\-]?[0-9]+)? |
[0-9]+([eE][\+\-]?[0-9]+)? {
#ifdef DEBUG
    ECHO;
#endif
    yylval = duplicate( yytext );
    return( REAL );
}

[0-9]+\.[0-9]*([dD][\+\-]?[0-9]+)? |
[0-9]*\.[0-9]+([dD][\+\-]?[0-9]+)? |
[0-9]+([dD][\+\-]?[0-9]+)? {
#ifdef DEBUG
    ECHO;
#endif
    yylval = duplicate( yytext );
    return( DOUBLE_PRECISION );
}

\'[^\']*\' |
\"[^\"]*\" {
#ifdef DEBUG
    ECHO;
#endif
    yytext[ 0 ] = '\\';
    yytext[ strlen( yytext ) - 1 ] = '\\';
    yylval = duplicate( yytext );
    return( STRING );
}

[0-9]+[hH] {
#ifdef DEBUG
    ECHO;
#endif
    yylval = duplicate( hollerith( yytext, '\\\"' ) );
    return( HOLLERITH );
}

```

**15. Appendix J: namelist program source**

FILE: namelist/Makefile

```

#
# Copyright 1991
# Georgia Institute of Technology
# Computer Engineering Research Laboratory
# Author: Stephen R. Wachtel
#

default:    namelist

CC = cc -g
INCLUDE = include
CFLAGS = -I$(INCLUDE)
LIBRARY = statement/statement.a library/library.a

OBJECTS = \
    $(INCLUDE)/grammar.h \
    *grammar.[co] \
    *scanner.[co] \
    yytrace.[co] \
    y.output

PROGRAMS = \
    *namelist

grammar.c: grammar.y
    yacc -dv grammar.y
    mv y.tab.h $(INCLUDE)/grammar.h
    mv y.tab.c grammar.c

scanner.c: scanner.l
    lex -vt scanner.l >scanner.c

scanner.o: scanner.c $(INCLUDE)/grammar.h
    $(CC) $(CFLAGS) -c scanner.c

grammar.o: grammar.c
    $(CC) $(CFLAGS) -c grammar.c

namelist: grammar.o scanner.o $(LIBRARY)
    $(CC) -o namelist grammar.o scanner.o $(LIBRARY)

sgrammar.c: grammar.c yytoken.awk
    awk -f yytoken.awk <grammar.c >sgrammar.c

sgrammar.o: sgrammar.c
    $(CC) $(CFLAGS) -c sgrammar.c

snamelist: sgrammar.o scanner.o $(LIBRARY)
    $(CC) -o snamelist sgrammar.o scanner.o $(LIBRARY)

dscanner.c: scanner.c
    cp scanner.c dscanner.c

dscanner.o: dscanner.c $(INCLUDE)/grammar.h
    $(CC) $(CFLAGS) -DDEBUG -c dscanner.c

dnamelist: grammar.o dscanner.o $(LIBRARY)
    $(CC) -o dnamelist grammar.o dscanner.o $(LIBRARY)

tgrammar.c: grammar.c
    sed 's/yystack:/& yytrace(yystate);/' <grammar.c >tgrammar.c

tgrammar.o: tgrammar.c
    $(CC) $(CFLAGS) -c tgrammar.c

```

```
tnamelist: tgrammar.o scanner.o yytrace.o $(LIBRARY)
$(CC) -o tnamelist tgrammar.o scanner.o yytrace.o $(LIBRARY)
```

```
yytrace.c: grammar.c yytrace.awk
awk -f yvtrace.awk <y.output >yytrace.c
```

```
yytrace.o: yytrace.c
$(CC) $(CFLAGS) -c yytrace.c
```

```
clean:
cd library; make clean
cd statement; make clean
rm -f $(PROGRAMS) $(OBJECTS)
```

```
FILE: namelist/grammar.y
```

```
/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */
```

```
%token CONTROL
%token DOUBLE PRECISION
%token IDENTIFIER
%token INTEGER
%token REAL
%token STRING
```

```
%right '='
%left '+' '-'
```

```
%{
typedef char *POINTER;
#define YYSTYPE POINTER

#include "namelist.h"
%}
```

```
%%
```

```
program:
    statement_list
    ;
```

```
statement_list:
    statement
    |
    statement_list statement
    ;
```

```
statement:
    control_statement
    |
    assignment_statement
    ;
```

```
control_statement:
    CONTROL
    {
        control_statement( $1 );
    }
    ;
```

```
assignment_statement:
    variable '=' constant_list
    {
```

```

        assignment_statement( $1, $3 );
    }
;

variable:
    IDENTIFIER
    {
        $$ = $1;
    }
    |
    IDENTIFIER '(' integer_list ')'
    {
        $$ = merge( "%s(%s)", $1, $3 );
        free( $1 );
        free( $3 );
    }
;

integer_list:
    INTEGER
    {
        $$ = $1;
    }
    |
    integer_list ',' INTEGER
    {
        $$ = merge( "%s,%s", $1, $3 );
        free( $1 );
        free( $3 );
    }
;

constant_list:
    Constant optional_comma
    {
        $$ = merge( "({%s})", $1 );
        free( $1 );
    }
    |
    constant_list constant optional_comma
    {
        $$ = merge( "%s{%s}", $1, $2 );
        free( $1 );
        free( $2 );
    }
;

optional_comma:
    /* NULL */
    |
    ','
;

constant:
    STRING
    {
        $$ = $1;
    }
    |
    signed_constant
    {
        $$ = number($1);
    }
    |
    INTEGER '*' signed_constant
    {
        $$ = merge( "%s*%s", $1, number($3) );
        free( $1 );
        free( $3 );
    }
;

signed_constant:
    unsigned_constant
    {

```

```

        $$ = $1;
    }
    |
    sign unsigned_constant
    {
        $$ = merge( "%s%s", $1, $2 );
        free( $1 );
        free( $2 );
    }
    ;

sign:
    '+'
    {
        $$ = duplicate( "+" );
    }
    |
    '-'
    {
        $$ = duplicate( "-" );
    }
    ;

unsigned_constant:
    INTEGER
    {
        $$ = $1;
    }
    |
    REAL
    {
        $$ = $1;
    }
    |
    DOUBLE_PRECISION
    {
        $$ = $1;
    }
    ;

%%

FILE: namelist/include/namelist.h

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

extern int count( );
extern char *duplicate( );
extern char *list( );
extern char *lowercase( );
extern char *merge( );
extern char *parse( );
extern char *number( );

FILE: namelist/library/Makefile

#
# Copyright 1991
# Georgia Institute of Technology
# Computer Engineering Research Laboratory
# Author: Stephen R. Wachtel
#

CC = cc -g
INCLUDE = ../include
CFLAGS = -I$(INCLUDE)
LIBRARY = library.a

```

```

OBJECTS = \
    count.o \
    duplicate.o \
    list.o \
    lowercase.o \
    main.o \
    margin_printf.o \
    number.o \
    merge.o \
    parse.o \
    yyerror.o \
    yywrap.o

$(LIBRARY):$(OBJECTS)
    ar crv $(LIBRARY) $(OBJECTS)
    ranlib $(LIBRARY)

.SUFFIXES: .c .o
.c.o:
    $(CC) -c $(CFLAGS) $<

clean:
    rm -f $(LIBRARY) $(OBJECTS)

FILE: namelist/library/count.c

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

int count( string, length, c )
register char *string;
register int length;
register char c;
{
    register int c_count = 0;

    while ( length != 0 )
    {
        if ( *string == c )
            c_count++;

        string++;
        length--;
    }

    return( c_count );
} /* count */

FILE: namelist/library/duplicate.c

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include <stdio.h>
#include <string.h>
#include <malloc.h>

char *duplicate( string )
register char *string;
{
    register char *temporary = (char *)NULL;

```

```

if ( string != (char *)NULL )
{
    if ( ( temporary = (char *)malloc( strlen( string ) + 1 ) ) != (char *)NULL )
        strcpy( temporary, string );
    else
        fprintf( stderr, "ERROR: duplicate( %s ,\n", string );
}

return( temporary );
} /* duplicate */

```

FILE: namelist/library/list.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```

extern char *parse( );
extern char *merge( );

```

```

char *list( input_list, delimiter )
register char *input_list;
register char *delimiter;
{
    register char *output_list;
    register char *list;
    register char *temporary;

    output_list = parse( input_list );
    list = parse( input_list );

    while ( list != (char *)0 )
    {
        temporary = merge( "%s%s%s", output_list, delimiter, list );

        free( output_list );
        free( list );

        output_list = temporary;
        list = parse( input_list );
    }

    return( output_list );
} /* list */

```

FILE: namelist/library/lowercase.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```

char *lowercase( string )
register char *string;
{
    register int index = 0;

    while ( string[ index ] != '\0' )
    {
        string[ index ] = tolower( string[ index ] );
        index++;
    }

    return( string );
} /* lowercase */

```

FILE: namelist/library/main.c



```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include <stdio.h>

extern FILE *yyin;
extern FILE *yyout;

#define PROGRAM argument[ 0 ]
#define INPUT_FILE argument[ 1 ]
#define OUTPUT_FILE argument[ 2 ]

int main( number_argument, argument )
int number_argument;
char *argument[ ];
{
    if ( number_argument == 1 )
    {
        yyin = stdin;
        yyout = stdout;

        yyparse( );
        exit( 0 );
    }

    if ( number_argument == 3 )
    {
        if ( ( yyin = fopen( INPUT_FILE, "r" ) ) == (FILE *)NULL )
        {
            fprintf( stderr, "%s: ERROR - unable to open input file '%s'\n", PROGRAM,
INPUT_FILE );
            exit( -1 );
        }

        if ( ( yyout = fopen( OUTPUT_FILE, "w" ) ) == (FILE *)NULL )
        {
            fprintf( stderr, "%s: ERROR - unable to open output file '%s'\n", PROGRAM,
OUTPUT_FILE );
            exit( -1 );
        }

        yyparse( );
        exit( 0 );
    }

    fprintf( stderr, "usage: %s <input file> <output file>\n", PROGRAM );
    exit( 0 );
} /* main */

```

FILE: namelist/library/margin\_printf.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include <stdio.h>
#include <string.h>

extern FILE *yyin;
extern FILE *yyout;

static void output_buffer( file, buffer )
register FILE *file;

```

```

register char *buffer;
{
#define LENGTH 72
    int length = LENGTH;
    int continuation = 0;
    int quote = 0;
    char temporary;

    while ( strlen( buffer ) > length )
    {
        if ( continuation++ != 0 )
            fprintf( file, "      &" );

        quote += count( buffer, length, '\\' );
        if ( ( quote % 2 ) == 0 )
        {
            while ( length != 0 )
            {
                if ( buffer[ length - 0 ] == '\\' )
                    break;

                if ( buffer[ length - 1 ] == '\\' )
                    break;

                length--;
            }

            if ( length == 0 )
            {
                fprintf( stderr, "ERROR: margin_printf()\n" );
                exit( -1 );
            }
        }

        temporary = buffer[ length ];
        buffer[ length ] = '\0';
        fprintf( file, "%s\n", buffer );
        buffer[ length ] = temporary;

        strcpy( &buffer[ 0 ], &buffer[ length ] );
        length = LENGTH - 6;
    }

    if ( strlen( buffer ) != 0 )
    {
        if ( continuation++ != 0 )
            fprintf( file, "      &" );

        fprintf( file, "%s\n", buffer );
    }
} /* output_buffer */

void margin_printf( buffer )
char *buffer;
{
    buffer[ strlen( buffer ) - 1 ] = '\0';
    while ( buffer[ strlen( buffer ) - 1 ] == ' ' )
        buffer[ strlen( buffer ) - 1 ] = '\0';

#ifdef PASSTHRU
    fprintf( yyout, "%s\n", buffer );
#else
    switch ( buffer[ 0 ] )
    {
        case '\0':
            fprintf( yyout, "\n" );
            break;

        case '*':
        case 'c':
        case 'C':
        case '$':
            fprintf( yyout, "%s\n", buffer );
            break;

        default:
            output_buffer( yyout, buffer );
    }
}
#endif

```

```

    buffer[ 0 ] = '\0';
} /* margin_printf */

```

```

FILE: namelist/library/merge.c

```

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```

#include <stdio.h>
#include <string.h>
#include <malloc.h>

```

```

#define STRLEN( s ) ( strlen( s ) - 2 )

```

```

char *merge( string, a, b, c, d )
register char *string;
register char *a;
register char *b;
register char *c;
register char *d;
{
    register char *temporary = (char *)NULL;

    switch ( count( string, strlen( string ), '%' ) )
    {
        case 0:
            if ( ( temporary = (char *)malloc( strlen( string ) + 1 ) ) != (char *)NULL )
                sprintf( temporary, string );
            else
                fprintf( stderr, "ERROR: merge( %s )\n", string );
            break;

        case 1:
            if ( ( temporary = (char *)malloc( strlen( string ) + STRLEN( a ) + 1 ) ) !=
(char *)NULL )
                sprintf( temporary, string, a );
            else
                fprintf( stderr, "ERROR: merge( %s, %s )\n", string, a );
            break;

        case 2:
            if ( ( temporary = (char *)malloc( strlen( string ) + STRLEN( a ) + STRLEN( b
) + 1 ) ) != (char *)NULL )
                sprintf( temporary, string, a, b );
            else
                fprintf( stderr, "ERROR: merge( %s, %s, %s )\n", string, a, b );
            break;

        case 3:
            if ( ( temporary = (char *)malloc( strlen( string ) + STRLEN( a ) + STRLEN( b
) + STRLEN( c ) + 1 ) ) != (char *)NULL )
                sprintf( temporary, string, a, b, c );
            else
                fprintf( stderr, "ERROR: merge( %s, %s, %s, %s )\n", string, a, b, c );
            break;

        case 4:
            if ( ( temporary = (char *)malloc( strlen( string ) + STRLEN( a ) + STRLEN( b
) + STRLEN( c ) + STRLEN( d ) + 1 ) ) != (char *)NULL )
                sprintf( temporary, string, a, b, c, d );
            else
                fprintf( stderr, "ERROR: merge( %s, %s, %s, %s, %s )\n", string, a, b, c, d
);
            break;

        default:
            fprintf( stderr, "ERROR: merge( %s )\n", string );
            break;
    }

    return( temporary );
} /* merge */

```

FILE: namelist/library/number.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include <string.h>

char *number( string )
register char *string;
{
    if ( strcmp( "+", string, 1 ) == 0 )
        strcpy( &string[0], &string[1] );

    if ( strchr( string, '.' ) != 0 )
    {
        while ( string[strlen(string)-1] == '0' )
            string[strlen(string)-1] = '\0';

        if ( strcmp( "-0.", string ) == 0 )
            strcpy( &string[0], &string[1] );
    }

    return( string );
} /* number */

```

FILE: namelist/library/parse.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include <string.h>

extern char *duplicate( );

char *parse( list )
register char *list;
{
    register int length = 0;
    register int brace = 0;
    register char *temporary = (char *)0;

    for (;;)
    {
        switch ( list[ length ] )
        {
            case '{':
                brace++;
                break;

            case '}':
                brace--;
                break;
        }

        if ( brace == 0 )
            break;

        length++;
    }

    if ( length != 0 )
    {
        list[ length ] = '\0';
    }
}

```

```

        temporary = duplicate( list + 1 );
        strcpy( list, list + 1 + length );
    }
    else
    {
        if ( list[ length ] != '\0' )
        {
            temporary = duplicate( list );
            list[ length ] = '\0';
        }
    }

    return( temporary );
} /* parse */

```

FILE: namelist/library/yyerror.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include <stdio.h>

extern int yylineno;

void yyerror( string )
register char *string;
{
    fprintf( stderr, "line %d, %s\n", yylineno, string );

    exit( -1 );
} /* yyerror */

```

FILE: namelist/library/yywrap.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

int yywrap( )
{
    return( 1 );
} /* yywrap */

```

FILE: namelist/scanner.l

```

%{
/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */
%}

a  [aA]
b  [bB]
c  [cC]
d  [dD]
e  [eE]
f  [fF]
g  [gG]
h  [hH]
i  [iI]

```

```

j  [jJ]
k  [kK]
l  [lL]
m  [mM]
n  [nN]
o  [oO]
p  [pP]
q  [qQ]
r  [rR]
s  [sS]
t  [tT]
u  [uU]
v  [vV]
w  [wW]
x  [xX]
y  [yY]
z  [zZ]

```

```

%{
#include "grammar.h"
extern char *yyval;

```

```

#include "namelist.h"
%}

```

```

%%

```

```

[\\] {
#ifdef DEBUG
    ECHO;
#endif
    /* return( ' ' ) */;
}

```

```

[\\n] {
#ifdef DEBUG
    ECHO;
#endif
    /* return( '\\n' ) */;
}

```

```

[\\t] {
#ifdef DEBUG
    ECHO;
#endif
    /* return( '\\t' ) */;
}

```

```

[\\+] {
#ifdef DEBUG
    ECHO;
#endif
    return( '+' );
}

```

```

[\\-] {
#ifdef DEBUG
    ECHO;
#endif
    return( '-' );
}

```

```

[\\()] {
#ifdef DEBUG
    ECHO;
#endif
    return( '(' );
}

```

```

[\\)] {
#ifdef DEBUG

```

```

    ECHO;
#endif
    return( ' ' );
}

[\\*] {
#ifdef DEBUG
    ECHO;
#endif
    return( '*' );
}

[\\,] {
#ifdef DEBUG
    ECHO;
#endif
    return( ',' );
}

[\\=] {
#ifdef DEBUG
    ECHO;
#endif
    return( '=' );
}

[a-zA-Z][_a-zA-Z0-9]* {
#ifdef DEBUG
    ECHO;
#endif
    yylval = duplicate( lowercase( yytext ) );
    return( IDENTIFIER );
}

[0-9]+ {
#ifdef DEBUG
    ECHO;
#endif
    yylval = duplicate( yytext );
    return( INTEGER );
}

[0-9]+\\. [0-9]* ([eE] [\\+\\-]? [0-9]+)? |
[0-9]*\\. [0-9]+ ([eE] [\\+\\-]? [0-9]+)? |
[0-9]+ ([eE] [\\+\\-]? [0-9]+)? {
#ifdef DEBUG
    ECHO;
#endif
    yylval = duplicate( yytext );
    return( REAL );
}

[0-9]+\\. [0-9]* ([dD] [\\+\\-]? [0-9]+)? |
[0-9]*\\. [0-9]+ ([dD] [\\+\\-]? [0-9]+)? |
[0-9]+ ([dD] [\\+\\-]? [0-9]+)? {
#ifdef DEBUG
    ECHO;
#endif
    yylval = duplicate( yytext );
    return( DOUBLE_PRECISION );
}

\\'[^\\']*\\' {
#ifdef DEBUG
    ECHO;
#endif
    yylval = duplicate( yytext );
    return( STRING );
}

$_a-zA-Z0-9]+ {
#ifdef DEBUG

```

```

    ECHO;
#endif
    yy1val = duplicate( yytext );
    return( CONTROL );
}

```

FILE: namelist/statement/Makefile

```

#
# Copyright 1991
# Georgia Institute of Technology
# Computer Engineering Research Laboratory
# Author: Stephen R. Wachtel
#

```

```

CC = cc -g
INCLUDE = ../include
CFLAGS = -I$(INCLUDE)
LIBRARY = statement.a

```

```

OBJECTS = \
    assignment_statement.o \
    control_statement.o

```

```

$(LIBRARY): $(OBJECTS)
    ar crv $(LIBRARY) $(OBJECTS)
    ranlib $(LIBRARY)

```

```

.SUFFIXES: .c .o
.c.o:
    $(CC) -c $(CFLAGS) $<

```

```

clean:
    rm -f $(LIBRARY) $(OBJECTS)

```

FILE: namelist/statement/assignment\_statement.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```

#include <stdio.h>
#include <string.h>

```

```

extern char *list( );

```

```

void assignment_statement( variable, constant_list )
register char *variable;
register char *constant_list;
{
#define BUFFER 4096
    char buffer[ BUFFER ];

    constant_list = list( constant_list, ", " );

    if ( ( strlen( variable ) + strlen( constant_list ) + 15 ) < BUFFER )
    {
        sprintf( buffer, "      DATA %s /%s/\n", variable, constant_list );
        margin_printf( buffer );
    }
    else
        fprintf( stderr, "ERROR: assignment_statement()\n" );

    free( variable );
    free( constant_list );
} /* assignment_statement */

```



FILE: namelist/statement/control\_statement.c

```
/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include <stdio.h>
#include <string.h>

void control_statement( string )
char *string;
{
#define BUFFER 4096
    char buffer[ BUFFER ];

    string[ 0 ] = '*';

    if ( ( strlen( string ) + 1 ) < BUFFER )
    {
        sprintf( buffer, "%s\n", string );
        margin_printf( buffer );
    }
    else
        fprintf( stderr, "ERROR: control_statement()\n" );

    free( string );
} /* control_statement */
```

**16. Appendix K: network program source**

FILE: network/communication/Makefile

```

#
# Copyright 1991
# Georgia Institute of Technology
# Computer Engineering Research Laboratory
# Author: Stephen R. Wachtel
#

default:    communication

CC = cc -g
INCLUDE = include
CFLAGS = -I$(INCLUDE)
LIBRARY = library/library.a

OBJECTS = \
    $(INCLUDE)/grammar.h \
    *grammar.[co] \
    *scanner.[co] \
    yytrace.[co] \
    y.output

PROGRAMS = \
    *communication

grammar.c: grammar.y
    yacc -dv grammar.y
    mv y.tab.h $(INCLUDE)/grammar.h
    mv y.tab.c grammar.c

scanner.c: scanner.l
    lex -vt scanner.l | sed 's/getc/yygetc/' >scanner.c

scanner.o: scanner.c $(INCLUDE)/grammar.h
    $(CC) $(CFLAGS) -c scanner.c

grammar.o: grammar.c
    $(CC) $(CFLAGS) -c grammar.c

communication: grammar.o scanner.o $(LIBRARY)
    $(CC) -o communication grammar.o scanner.o $(LIBRARY)

sgrammar.c: grammar.c yytoken.awk
    awk -f yytoken.awk <grammar.c >sgrammar.c

sgrammar.o: sgrammar.c
    $(CC) $(CFLAGS) -c sgrammar.c

scommunication:    sgrammar.o scanner.o $(LIBRARY)
    $(CC) -o scommunication sgrammar.o scanner.o $(LIBRARY)

dscanner.c: scanner.c
    cp scanner.c dscanner.c

dscanner.o: dscanner.c $(INCLUDE)/grammar.h
    $(CC) $(CFLAGS) -DDEBUG -c dscanner.c

dcommunication:    grammar.o dscanner.o $(LIBRARY)
    $(CC) -o dcommunication grammar.o dscanner.o $(LIBRARY)

tgrammar.c: grammar.c
    sed 's/yystack:/& yytrace(yystate);/' <grammar.c >tgrammar.c

tgrammar.o: tgrammar.c
    $(CC) $(CFLAGS) -c tgrammar.c

```

```
tcommunication:  tgrammar.o scanner.o yytrace.o $(LIBRARY)
                  $(CC) -o tcommunication tgrammar.o scanner.o yytrace.o $(LIBRARY)
```

```
yytrace.c: grammar.c yytrace.awk
            awk -f yytrace.awk <y.output >yytrace.c
```

```
yytrace.o: yytrace.c
            $(CC) $(CFLAGS) -c yytrace.c
```

```
clean:
    cd library; make clean
    rm -f $(PROGRAMS) $(OBJECTS)
```

```
FILE: network/communication/grammar.y
```

```
/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */
```

```
/*
 * FORTRAN 77
 */
```

```
%token RW_AND
%token RW_ASSIGN
%token RW_BACKSPACE
%token RW_BLOCK_DATA
%token RW_CALL
%token RW_CHARACTER
%token RW_CLOSE
%token RW_COMMON
%token RW_COMPLEX
%token RW_CONTINUE
%token RW_DATA
%token RW_DIMENSION
%token RW_DO
%token RW_DOUBLE_PRECISION
%token RW_ELSE
%token RW_ELSE_IF
%token RW_END
%token RW_END IF
%token RW_ENDFILE
%token RW_ENTRY
%token RW_EQ
%token RW_EQUIVALENCE
%token RW_EQV
%token RW_EXTERNAL
%token RW_FALSE
%token RW_FORMAT
%token RW_FUNCTION
%token RW_GE
%token RW_GO_TO
%token RW_GT
%token RW_IF
%token RW_IMPLICIT
%token RW_INCLUDE
%token RW_INQUIRE
%token RW_INTEGER
%token RW_INTRINSIC
%token RW_LE
%token RW_LOGICAL
%token RW_LT
%token RW_NAMELIST
%token RW_NE
%token RW_NEQV
%token RW_NOT
%token RW_OPEN
%token RW_OR
%token RW_PARAMETER
%token RW_PAUSE
%token RW_PRINT
%token RW_PROGRAM
```

```

%token RW_READ
%token RW_REAL
%token RW_RETURN
%token RW_REWIND
%token RW_SAVE
%token RW_STOP
%token RW_SUBROUTINE
%token RW_THEN
%token RW_TO
%token RW_TRUE
%token RW_WRITE
%token RW_UNDEFINED

%token COMMENT
%token CONCATENATE
%token DOUBLE_PRECISION
%token EXPONENTIATE
%token HOLLERITH
%token IDENTIFIER
%token INTEGER
%token LABEL
%token REAL
%token STRING

%left ','
%nonassoc ':'
%right '='
%left RW_EQV RW_NEQV
%left RW_OR
%left RW_AND
%left RW_NOT
%nonassoc RW_EQ RW_NE RW_LT RW_LE RW_GT RW_GE
%left CONCATENATE
%left '+' '-'
%left '*' '/'
%right EXPONENTIATE
%left SIGN

%{
typedef char *POINTER;
#define YYSTYPE POINTER

#include "list.h"
static LIST *block_list = 0;
static LIST *call_list;

static POINTER block_name;
static POINTER block_type;

extern POINTER array( );
extern POINTER duplicate( );
extern POINTER merge( );
%}

%%

program:
    optional_statement_list
    {
        summary( block_list );
    }
    ;

optional_statement_list:
    /* NULL */
    |
        statement_list
    ;

statement_list:
    statement
    |
        statement_list statement
    ;

```

```

statement:
    comment_statement
    |
    label unlabeled_statement
    ;

comment_statement:
    COMMENT
    ;

label:
    LABEL
    ;

unlabeled_statement:
    include_statement
    |
    program_statement
    |
    block_data_statement
    |
    function_statement
    |
    subroutine_statement
    |
    entry_statement
    |
    end_statement
    |
    specification_statement
    |
    executable_statement
    |
    format_statement
    ;

include_statement:
    RW_INCLUDE character_constant
    ;

program_statement:
    RW_PROGRAM program_identifier
    ;

program_identifier:
    IDENTIFIER
    {
        block_name = $1;
        block_type = duplicate( "+PROGRAM" );
        call_list = 0;
    }
    ;

block_data_statement:
    RW_BLOCK_DATA block_data_identifier
    ;

block_data_identifier:
    IDENTIFIER
    {
        block_name = $1;
        block_type = duplicate( "+BLOCK_DATA" );
        call_list = 0;
    }
    ;

function_statement:
    RW_FUNCTION function_identifier optional_formal_argument_list
    |
    type RW_FUNCTION function_identifier optional_formal_argument_list

```

```

;

function_identifier:
    IDENTIFIER
    {
        block_name = $1;
        block_type = duplicate( "+FUNCTION" );
        call_list = 0;
    }
;

subroutine_statement:
    RW_SUBROUTINE subroutine_identifier
    |
    RW_SUBROUTINE subroutine_identifier optional_formal_argument_list
;

subroutine_identifier:
    IDENTIFIER
    {
        block_name = $1;
        block_type = duplicate( "+SUBROUTINE" );
        call_list = 0;
    }
;

entry_statement:
    RW_ENTRY entry_identifier
    |
    RW_ENTRY entry_identifier optional_formal_argument_list
;

entry_identifier:
    IDENTIFIER
;

optional_formal_argument_list:
    '(' ',' ')'
    |
    '(' formal_argument_list ')'
;

formal_argument_list:
    formal_argument
    |
    formal_argument_list ',' formal_argument
;

formal_argument:
    IDENTIFIER
    |
    formal_argument_alternate_return
;

formal_argument_alternate_return:
    '*'
;

end_statement:
    RW_END
    {
        add_list( &block_list, block_name, block_type, call_list );
    }
;

specification_statement:
    external_statement
    |
    intrinsic_statement
;

```

```

parameter_statement
|
dimension_statement
|
declaration_statement
|
save_statement
|
common_statement
|
equivalence_statement
|
implicit_statement
|
data_statement
|
namelist_statement
;

external_statement:
    RW_EXTERNAL external_list
;

external_list:
    external
|
    external_list ',' external
;

external:
    IDENTIFIER
;

intrinsic_statement:
    RW_INTRINSIC intrinsic_list
;

intrinsic_list:
    intrinsic
|
    intrinsic_list ',' intrinsic
;

intrinsic:
    IDENTIFIER
;

parameter_statement:
    RW_PARAMETER '(' parameter_list ')'
;

parameter_list:
    parameter
|
    parameter_list ',' parameter
;

parameter:
    IDENTIFIER '=' expression
;

dimension_statement:
    RW_DIMENSION dimension_list
;

dimension_list:
    dimension
|
    dimension_list ',' dimension
;

```

```

dimension:
    IDENTIFIER '(' subscript_list ')'
    ;

subscript_list:
    subscript
    |
    subscript_list ',' subscript
    ;

subscript:
    upper_bound
    |
    lower_bound ':' upper_bound
    ;

lower_bound:
    expression
    ;

upper_bound:
    lower_bound
    |
    upper_bound_adjustable
    ;

upper_bound_adjustable:
    '*'
    ;

declaration_statement:
    type declaration_list
    ;

declaration_list:
    declaration
    |
    declaration_list ',' declaration
    ;

declaration..
    IDENTIFIER optional_type_length
    |
    IDENTIFIER '(' subscript_list ')' optional_type_length
    ;

type:
    type_name optional_type_length
    ;

type_name:
    RW_CHARACTER
    |
    RW_COMPLEX
    |
    RW_DOUBLE_PRECISION
    |
    RW_INTEGER
    |
    RW_LOGICAL
    |
    RW_REAL
    |
    RW_UNDEFINED
    ;

optional_type_length:
    /* NULL */

```



```

|
|   type_length
;

type_length:
    '*' INTEGER
|
|   '*' type_length_adjustable
;

type_length_adjustable:
    '(' '*' ')'
;

save_statement:
    RW_SAVE optional_save_list
;

optional_save_list:
    /* NULL */
|
|   save_list
;

save_list:
    save
|
|   save_list ',' save
;

save:
    IDENTIFIER
|
|   common_name
;

common_statement:
    RW_COMMON optional_common_name common_list
;

optional_common_name:
    /* NULL */
|
|   common_name
;

common_name:
    '/' optional_identifier '/'
;

optional_identifier:
    /* NULL */
|
|   IDENTIFIER
;

common_list:
    common
|
|   common_list ',' common
;

common:
    IDENTIFIER
|
|   IDENTIFIER '(' subscript_list ')'
;

```

```
equivalence_statement:
    RW_EQUIVALENCE equivalence_list
    ;

equivalence_list:
    equivalence
    |
    equivalence_list ',' equivalence
    ;

equivalence:
    '(' variable_list ')'
    ;

variable_list:
    variable
    |
    variable_list ',' variable
    ;

implicit_statement:
    RW_IMPLICIT type '(' implicit_list ')'
    ;

implicit_list:
    implicit
    |
    implicit_list ',' implicit
    ;

implicit:
    IDENTIFIER
    |
    IDENTIFIER '-' IDENTIFIER
    ;

namelist_statement:
    RW_NAMELIST namelist_name namelist_list
    ;

namelist_name:
    '/' IDENTIFIER '/'
    ;

namelist_list:
    namelist
    |
    namelist_list ',' namelist
    ;

namelist:
    IDENTIFIER
    ;

data_statement:
    RW_DATA data_list
    ;

data_list:
    data
    |
    data_list optional_comma data
    ;

data:
    data_variable_list '/' data_constant_list '/'
    ;
```

```

data_variable_list:
    data_variable
    |
    data_variable_list ',' data_variable
    ;

data_variable:
    variable
    |
    data_implied_do_list
    ;

data_implied_do_list:
    '(' data_variable_list ',' IDENTIFIER '=' expression_list ')'
    ;

data_constant_list:
    data_constant
    |
    data_constant_list ',' data_constant
    ;

data_constant:
    data_initialization
    |
    IDENTIFIER '*' data_initialization
    |
    INTEGER '*' data_initialization
    ;

data_initialization:
    IDENTIFIER
    |
    character_constant
    |
    logical_constant
    |
    signed_numerical_constant
    ;

signed_numerical_constant:
    numerical_constant
    |
    '+' numerical_constant %prec SIGN
    |
    '-' numerical_constant %prec SIGN
    ;

expression:
    parenthesis_expression
    {
        $$ = $1;
    }
    |
    simple_expression
    {
        $$ = $1;
    }
    ;

parenthesis_expression:
    '(' expression ')'
    {
        $$ = 0;
    }
    ;

simple_expression:
    variable
    {
        $$ = $1;
    }

```

```

    }
    |
    constant
    {
        $$ = $1;
    }
    |
    arithmetic_expression
    {
        $$ = $1;
    }
    |
    character_expression
    {
        $$ = $1;
    }
    |
    relational_expression
    {
        $$ = $1;
    }
    |
    logical_expression
    {
        $$ = $1;
    }
    |
    unary_expression
    {
        $$ = $1;
    }
    ;

variable:
    IDENTIFIER
    {
        $$ = $1;
    }
    |
    IDENTIFIER string_subset
    {
        $$ = merge( "%s%s", $1, $2 );
    }
    |
    array
    {
        $$ = $1;
    }
    ;

array:
    IDENTIFIER '(' optional_expression_list ')'
    {
        $$ = array( $1, $3 );
    }
    |
    IDENTIFIER '(' optional_expression_list ')' string_subset
    {
        $$ = merge( "%s%s", array( $1, $3 ), $5 );
    }
    ;

optional_expression_list:
    /* NULL */
    {
        $$ = 0;
    }
    |
    expression_list
    {
        $$ = $1;
    }
    ;

expression_list:
    expression
    {

```

```

    $$ = merge( "%s)", $1 );
}
|
expression_list ',' expression
{
    $$ = merge( "%s%s)", $1, $3 );
}
;

string_subset:
'(' optional_expression ':' optional_expression ')'
{
    $$ = merge( "%s:%s)", $2, $4 );
}
;

optional_expression:
/* NULL */
{
    $$ = 0;
}
|
expression
{
    $$ = $1;
}
;

constant:
character_constant
{
    $$ = $1;
}
|
logical_constant
{
    $$ = $1;
}
|
numerical_constant
{
    $$ = $1;
}
;

character_constant:
HOLLERITH
{
    $$ = $1;
}
|
STRING
{
    $$ = $1;
}
;

logical_constant:
RW_FALSE
{
    $$ = duplicate( ".FALSE." );
}
|
RW_TRUE
{
    $$ = duplicate( ".TRUE." );
}
;

numerical_constant:
DOUBLE_PRECISION
{
    $$ = $1;
}
|

```

```

    INTEGER
    {
        $$ = $1;
    }
|
    REAL
    {
        $$ = $1;
    }
;

arithmetic_expression:
    expression '+' expression %prec '+'
    {
        $$ = 0;
    }
|
    expression '-' expression %prec '-'
    {
        $$ = 0;
    }
|
    expression '*' expression %prec '*'
    {
        $$ = 0;
    }
|
    expression '/' expression %prec '/'
    {
        $$ = 0;
    }
|
    expression EXPONENTIATE expression %prec EXPONENTIATE
    {
        $$ = 0;
    }
;

character_expression:
    expression '/' '/' expression %prec CONCATENATE
    {
        $$ = 0;
    }
;

relational_expression:
    expression RW_EQ expression %prec RW_EQ
    {
        $$ = 0;
    }
|
    expression RW_NE expression %prec RW_NE
    {
        $$ = 0;
    }
|
    expression RW_LT expression %prec RW_LT
    {
        $$ = 0;
    }
|
    expression RW_LE expression %prec RW_LE
    {
        $$ = 0;
    }
|
    expression RW_GT expression %prec RW_GT
    {
        $$ = 0;
    }
|
    expression RW_GE expression %prec RW_GE
    {
        $$ = 0;
    }
;

```

```

logical_expression:
    expression RW_AND expression %prec RW_AND
    {
        $$ = 0;
    }
    |
    expression RW_OR expression %prec RW_OR
    {
        $$ = 0;
    }
    |
    expression RW_EQV expression %prec RW_EQV
    {
        $$ = 0;
    }
    |
    expression RW_NEQV expression %prec RW_NEQV
    {
        $$ = 0;
    }
    ;

```

```

unary_expression:
    '+' expression %prec SIGN
    {
        $$ = merge( "+%s", $2 );
    }
    |
    '-' expression %prec SIGN
    {
        $$ = merge( "-%s", $2 );
    }
    |
    RW_NOT expression %prec RW_NOT
    {
        $$ = merge( ".NOT.%s", $2 );
    }
    ;

```

```

executable_statement:
    do_statement
    |
    logical_if_statement
    |
    block_if_statement
    |
    else_statement
    |
    else_if_statement
    |
    end_if_statement
    |
    subset_executable_statement
    ;

```

```

do_statement:
    RW_DO INTEGER IDENTIFIER '=' expression_list
    ;

```

```

logical_if_statement:
    if_expression subset_executable_statement
    ;

```

```

if_expression:
    RW_IF '(' expression ')'
    ;

```

```

block_if_statement:
    RW_IF '(' expression ')' RW_THEN
    ;

```

```

else_statement:
    RW_ELSE
    ;

```

```

else_if_statement:
    RW_ELSE_IF '(' expression ')' RW_THEN
    ;

end_if_statement:
    RW_END_IF
    ;

subset_executable_statement:
    assignment_statement
    |
    assign_statement
    |
    arithmetic_if_statement
    |
    continue_statement
    |
    call_statement
    |
    return_statement
    |
    unconditional_go_to_statement
    |
    computed_go_to_statement
    |
    assigned_go_to_statement
    |
    stop_statement
    |
    pause_statement
    |
    io_statement
    ;

assignment_statement:
    variable '=' expression
    ;

assign_statement:
    RW_ASSIGN INTEGER RW_TO IDENTIFIER
    ;

arithmetic_if_statement:
    RW_IF '(' expression ')' integer_list
    ;

continue_statement:
    RW_CONTINUE
    ;

call_statement:
    RW_CALL IDENTIFIER
    {
        add_list( &call_list, $2, 0, 0 );
    }
    |
    RW_CALL IDENTIFIER optional_actual_argument_list
    {
        if ( ( strcmp( $2, "SEND ", 5 ) == 0 )
            || ( strcmp( $2, "RECEIVE ", 8 ) == 0 ) )
            add_list( &call_list, $2, $3, 0 );
        else
            add_list( &call_list, $2, 0, 0 );
    }
    ;

optional_actual_argument_list:
    '(' ')'
    {
        $$ = 0;
    }

```



```

|
|   '(' actual_argument_list ')'
|   {
|       $$ = $2;
|   }
;

actual_argument_list:
    actual_argument
    {
        $$ = $1;
    }
|
|   actual_argument_list ',' actual_argument
|   {
|       $$ = 0;
|   }
;

actual_argument:
    expression
    {
        $$ = $1;
    }
|
|   actual_argument_alterate_return
|   {
|       $$ = 0;
|   }
;

actual_argument_alterate_return:
    '*' INTEGER
;

return_statement:
    RW_RETURN optional_expression
;

unconditional_go_to_statement:
    RW_GO_TO INTEGER
;

computed_go_to_statement:
    RW_GO_TO '(' integer_list ')' optional_comma expression
;

assigned_go_to_statement:
    RW_GO_TO IDENTIFIER
|
|   RW_GO_TO IDENTIFIER optional_comma '(' integer_list ')'
;

optional_comma:
    /* NULL */
|
|   ','
;

integer_list:
    INTEGER
|
|   integer_list ',' INTEGER
;

pause_statement:
    RW_PAUSE optional_expression
;

stop_statement:

```

```

        RW_STOP optional_expression
    ;

io_statement:
    open_statement
    |
    close_statement
    |
    inquire_statement
    |
    read_statement
    |
    write_statement
    |
    print_statement
    |
    backspace_statement
    |
    rewind_statement
    |
    endfile_statement
    ;

open_statement:
    RW_OPEN '(' control_information_list ')'
    ;

close_statement:
    RW_CLOSE '(' control_information_list ')'
    ;

inquire_statement:
    RW_INQUIRE '(' control_information_list ')'
    ;

read_statement:
    RW_READ '(' control_information_list ')' optional_io_list
    |
    RW_READ control
    |
    RW_READ control ',' io_list
    ;

write_statement:
    RW_WRITE '(' control_information_list ')' optional_io_list
    ;

print_statement:
    RW_PRINT control
    |
    RW_PRINT control ',' io_list
    ;

backspace_statement:
    RW_BACKSPACE '(' control_information_list ')'
    |
    RW_BACKSPACE control
    ;

rewind_statement:
    RW_REWIND '(' control_information_list ')'
    |
    RW_REWIND control
    ;

endfile_statement:
    RW_ENDFILE '(' control_information_list ')'
    |
    RW_ENDFILE control
    ;

```

```

control_information_list:
    control_information
    |
    control_information_list ',' control_information
    ;

control_information:
    control
    |
    IDENTIFIER '=' expression
    ;

control:
    variable
    |
    constant
    |
    '*'
    ;

optional_io_list:
    /* NULL */
    |
    io_list
    ;

io_list:
    io
    |
    io_list ',' io
    ;

io:
    expression
    |
    io_implied_do_list
    ;

io_implied_do_list:
    '(' io_list ',' IDENTIFIER '=' expression_list ')'
    ;

format_statement:
    RW_FORMAT
    ;

%%

FILE: network/communication/include/list.h

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#define LIST struct list_type
LIST
{
    char *identifier;
    char *argument;
    LIST *call_list;
    LIST *next;
};

extern LIST *end_list( );
extern LIST *add_list( );

```

```
extern LIST *find_list( );
extern void print_list( );
extern void delete_list( );
```

FILE: network/communication/library/Makefile

```
#
# Copyright 1991
# Georgia Institute of Technology
# Computer Engineering Research Laboratory
# Author: Stephen R. Wachtel
#
```

```
CC = cc -g
INCLUDE = ../include
CFLAGS = -I$(INCLUDE)
LIBRARY = library.a
```

```
OBJECTS = \
    array.o \
    count.o \
    duplicate.o \
    hollerith.o \
    link_list.o \
    list.o \
    main.o \
    merge.o \
    non_blank.o \
    parse.o \
    summary.o \
    uppercase.o \
    yyerror.o \
    yygetc.o \
    yywrap.o
```

```
$(LIBRARY): $(OBJECTS)
    ar crv $(LIBRARY) $(OBJECTS)
    ranlib $(LIBRARY)
```

```
.SUFFIXES: .c .o
.c.o:
    $(CC) -c $(CFLAGS) $<
```

```
clean:
    rm -f $(LIBRARY) $(OBJECTS)
```

FILE: network/communication/library/array.c

```
/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */
```

```
extern char *list( );
extern char *merge( );
```

```
char *array( identifier, optional_expression_list )
register char *identifier;
register char *optional_expression_list;
{
    if ( optional_expression_list != (char *)0 )
        return( merge( "%s(%s)", identifier, list( optional_expression_list, ", " ) ) );
    else
        return( merge( "%s()", identifier ) );
} /* array */
```

FILE: network/communication/library/count.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```

int count( string, length, c )
register char *string;
register int length;
register char c;
{
    register int c_count = 0;

    while ( length != 0 )
    {
        if ( *string == c )
            c_count++;

        string++;
        length--;
    }

    return( c_count );
} /* count */

```

FILE: network/communication/library/duplicate.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```

#include <stdio.h>
#include <string.h>
#include <malloc.h>

```

```

char *duplicate( string )
register char *string;
{
    register char *temporary = (char *)NULL;

    if ( string != (char *)NULL )
    {
        if ( ( temporary = (char *)malloc( strlen( string ) + 1 ) ) != (char *)NULL )
            strcpy( temporary, string );
        else
            fprintf( stderr, "ERROR: duplicate( %s )\n", string );
    }

    return( temporary );
} /* duplicate */

```

FILE: network/communication/library/hollerith.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```

#include <stdio.h>

```

```

char *hollerith( string, delimiter )
register char *string;
register char delimiter;
{

```

```

int hollerith_length;
register int string_length = 0;

sscanf( string, "%dh", &hollerith_length );

string[ string_length++ ] = delimiter;
while ( hollerith_length != 0 )
{
    if ( ( string[ string_length ] = yyinput( ) ) == '\n' )
    {
        yyunput( string[ string_length ] );
        break;
    }

    string_length++;
    hollerith_length--;
}
string[ string_length++ ] = delimiter;

string[ string_length ] = '\0';
return( string );
} /* hollerith */

```

FILE: network/communication/library/link\_list.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```

#include <stdio.h>
#include <malloc.h>
#include <string.h>
#include "list.h"

```

```

LIST *end_list( list )
register LIST *list;
{
    if ( list != (LIST *)NULL )
    {
        while ( list->next != (LIST *)NULL )
            list = list->next;
    }

    return( list );
} /* end_list */

```

```

LIST *add_list( list, identifier, argument, call_list )
register LIST **list;
register char *identifier;
register char *argument;
register LIST *call_list;
{
    register LIST *temporary = (LIST *)malloc( sizeof( LIST ) );

    temporary->identifier = identifier;
    temporary->argument = argument;
    temporary->call_list = call_list;
    temporary->next = (LIST *)NULL;

    if ( *list == (LIST *)NULL )
        *list = temporary;
    else
        end_list( *list )->next = temporary;

    return( temporary );
} /* add_list */

```

```

LIST *find_list( list, identifier )
register LIST *list;
register char *identifier;
{
    while ( list != (LIST *)NULL )

```

```

    {
        if ( strcmp( list->identifier, identifier ) == 0 )
            return( list );

        list = list->next;
    }

    return( (LIST *)NULL );
} /* find_list */

void print_list( file, list )
register FILE *file;
register LIST *list;
{
    register LIST *call_list;

    while ( list != (LIST *)NULL )
    {
        fprintf( file, "%s %s\n", list->identifier, list->argument );

        call_list = list->call_list;
        while ( call_list != (LIST *)NULL )
        {
            fprintf( file, "\t%s", call_list->identifier );
            if ( call_list->argument != 0 )
                fprintf( file, "( %s )", call_list->argument );
            fprintf( file, "\n" );

            call_list = call_list->next;
        }
        fprintf( file, "\n" );

        list = list->next;
    }
} /* print_list */

void delete_list( list )
register LIST *list;
{
    if ( list != (LIST *)NULL )
    {
        delete_list( list->next );

        free( list );
    }
} /* delete_list */

```

FILE: network/communication/library/list.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```

extern char *parse( );
extern char *merge( );

```

```

char *list( input_list, delimiter )
register char *input_list;
register char *delimiter;
{
    register char *output_list;
    register char *list;
    register char *temporary;

    output_list = parse( input_list );
    list = parse( input_list );

    while ( list != (char *)0 )
    {
        temporary = merge( "%s%s", output_list, delimiter, list );

        free( output_list );
    }
}

```

```

        free( list );

        output_list = temporary;
        list = parse( input_list );
    }

    return( output_list );
} /* list */

```

FILE: network/communication/library/main.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include <stdio.h>

extern FILE *yyin;
extern FILE *yyout;

#define PROGRAM argument[ 0 ]
#define INPUT_FILE argument[ 1 ]
#define OUTPUT_FILE argument[ 2 ]

int main( number_argument, argument )
int number_argument;
char *argument[ ];
{
    if ( number_argument == 1 )
    {
        yyin = stdin;
        yyout = stdout;

        yyparse( );
        exit( 0 );
    }

    if ( number_argument == 3 )
    {
        if ( ( yyin = fopen( INPUT_FILE, "r" ) ) == (FILE *)NULL )
        {
            fprintf( stderr, "%s: ERROR - unable to open input file '%s'\n", PROGRAM,
INPUT_FILE );
            exit( -1 );
        }

        if ( ( yyout = fopen( OUTPUT_FILE, "w" ) ) == (FILE *)NULL )
        {
            fprintf( stderr, "%s: ERROR - unable to open output file '%s'\n", PROGRAM,
OUTPUT_FILE );
            exit( -1 );
        }

        yyparse( );
        exit( 0 );
    }

    fprintf( stderr, "usage: %s <input file> <output file>\n", PROGRAM );
    exit( 0 );
} /* main */

```

FILE: network/communication/library/merge.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```



```

#include <stdio.h>
#include <string.h>
#include <malloc.h>

#define STRLEN( s ) ( strlen( s ) - 2 )

char *merge( string, a, b, c, d )
register char *string;
register char *a;
register char *b;
register char *c;
register char *d;
{
    register char *temporary = (char *)NULL;

    switch ( count( string, strlen( string ), '%' ) )
    {
        case 0:
            if ( ( temporary = (char *)malloc( strlen( string ) + 1 ) ) != (char *)NULL )
                sprintf( temporary, string );
            else
                fprintf( stderr, "ERROR: merge( %s )\n", string );
            break;

        case 1:
            if ( ( temporary = (char *)malloc( strlen( string ) + STRLEN( a ) + 1 ) ) !=
                (char *)NULL )
                sprintf( temporary, string, a );
            else
                fprintf( stderr, "ERROR: merge( %s, %s )\n", string, a );
            break;

        case 2:
            if ( ( temporary = (char *)malloc( strlen( string ) + STRLEN( a ) + STRLEN( b )
                + 1 ) ) != (char *)NULL )
                sprintf( temporary, string, a, b );
            else
                fprintf( stderr, "ERROR: merge( %s, %s, %s )\n", string, a, b );
            break;

        case 3:
            if ( ( temporary = (char *)malloc( strlen( string ) + STRLEN( a ) + STRLEN( b )
                + STRLEN( c ) + 1 ) ) != (char *)NULL )
                sprintf( temporary, string, a, b, c );
            else
                fprintf( stderr, "ERROR: merge( %s, %s, %s, %s )\n", string, a, b, c );
            break;

        case 4:
            if ( ( temporary = (char *)malloc( strlen( string ) + STRLEN( a ) + STRLEN( b )
                + STRLEN( c ) + STRLEN( d ) + 1 ) ) != (char *)NULL )
                sprintf( temporary, string, a, b, c, d );
            else
                fprintf( stderr, "ERROR: merge( %s, %s, %s, %s, %s )\n", string, a, b, c, d );
            break;

        default:
            fprintf( stderr, "ERROR: merge( %s )\n", string );
            break;
    }

    return( temporary );
} /* merge */

```

FILE: network/communication/library/non\_blank.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```

#include <string.h>

```

```

char *non_blank( string )
register char *string;
{
    register int offset;
    register int length;

    length = strlen( string ) - 1;
    while ( ( string[ length ] == ' ' ) && ( string[ length ] != '\0' ) )
        string[ length-- ] = '\0';

    offset = 0;
    while ( ( string[ offset ] == ' ' ) && ( string[ offset ] != '\0' ) )
        string[ offset++ ] = '\0';

    strcpy( string, &string[ offset ] );

    if ( strlen( string ) != 0 )
        return( string );
    else
        return( 0 );
} /* non_blank */

```

FILE: network/communication/library/parse.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```
#include <string.h>
```

```
extern char *duplicate( );
```

```

char *parse( list )
register char *list;
{
    register int length = 0;
    register int brace = 0;
    register char *temporary = (char *)0;

    for (;;)
    {
        switch ( list[ length ] )
        {
            case '(':
                brace++;
                break;

            case ')':
                brace--;
                break;

            }

        if ( brace == 0 )
            break;

        length++;
    }

    if ( length != 0 )
    {
        list[ length ] = '\0';
        temporary = duplicate( list + 1 );
        strcpy( list, list + 1 + length );
    }
    else
    {
        if ( list[ length ] != '\0' )
        {
            temporary = duplicate( list );
            list[ length ] = '\0';
        }
    }
}

```

```

    }

    return( temporary );
} /* parse */

```

FILE: network/communication/library/summary.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include <stdio.h>
#include <string.h>
#include "list.h"

extern FILE *yyin;
extern FILE *yyout;

#ifdef DEBUG
void print_level( level )
register int level;
{
    while ( level-- != 0 )
        fprintf( yyout, "\t" );
} /* print_level */

void print_trace( block_list, identifier, argument, level )
register LIST *block_list;
register char *identifier;
register char *argument;
register int level;
{
    LIST *list;
    LIST *call_list;

    print_level( level );
    fprintf( yyout, "%s", identifier );

    if ( argument != (char *)NULL )
        fprintf( yyout, "( %s )", argument );
    fprintf( yyout, "\n" );

    if ( ( list = find_list( block_list, identifier ) ) == (LIST*)NULL )
        return;

    if ( *list->argument == '+' )
    {
        *list->argument = '-';

        call_list = list->call_list;
        while ( call_list != (LIST *)NULL )
        {
            print_trace( block_list, call_list->identifier, call_list->argument, level + 1 );

            call_list = call_list->next;
        }

        *list->argument = '+';
    }
} /* print_trace */
#else
void print_trace( block_list, identifier, argument, level )
register LIST *block_list;
register char *identifier;
register char *argument;
register int level;
{
    LIST *list;
    LIST *call_list;

    if ( strncmp( identifier, "SEND_", 5 ) == 0 )

```

```

    {
        fprintf( yyout, "%s %s S\n", argument, &identifier[ 5 ] );
        return;
    }

    if ( strcmp( identifier, "RECEIVE_", 8 ) == 0 )
    {
        fprintf( yyout, "%s %s R\n", argument, &identifier[ 8 ] );
        return;
    }

    if ( ( list = find_list( block_list, identifier ) ) == (LIST *)NULL )
        return;

    if ( *list->argument == '+' )
    {
        *list->argument = '-';

        call_list = list->call_list;
        while ( call_list != (LIST *)NULL )
        {
            print_trace( block_list, call_list->identifier, call_list->argument, level + 1 );

            call_list = call_list->next;
        }

        *list->argument = '+';
    }
} /* print_trace */
#endif

```

```

void summary( list )
register LIST *list;
{
    while ( list != (LIST *)NULL )
    {
        if ( strcmp( list->argument, "+PROGRAM" ) == 0 )
            print_trace( list, list->identifier, 0, 0 );

        list = list->next;
    }
} /* summary */

```

FILE: network/communication/library/uppercase.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```

char *uppercase( string )
register char *string;
{
    register int index = 0;

    while ( string[ index ] != '\0' )
    {
        string[ index ] = toupper( string[ index ] );
        index++;
    }

    return( string );
} /* uppercase */

```

FILE: network/communication/library/yyerror.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```
#include <stdio.h>

extern int yylineno;

void yyerror( string )
register char *string;
{
    fprintf( stderr, "line %d, %s\n", yylineno, string );
    exit( -1 );
} /* yyerror */
```

FILE: network/communication/library/yygetc.c

```
/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include <stdio.h>
#include <ctype.h>

extern int yylineno;

int tab( length )
register int length;
{
    while ( length-- != 0 )
        yyunput( ' ' );

    return( ' ' );
} /* tab */

int yygetc( file )
register FILE *file;
{
    int c;
    int column[ 6 ];

loop:
    if ( ( c = getc( file ) ) == '\t' )
        c = tab( 6 );

    if ( c != '\n' )
        return( c );

    if ( ( column[ 0 ] = getc( file ) ) != ' ' )
        goto abort_0;
    if ( ( column[ 1 ] = getc( file ) ) != ' ' )
        goto abort_1;
    if ( ( column[ 2 ] = getc( file ) ) != ' ' )
        goto abort_2;
    if ( ( column[ 3 ] = getc( file ) ) != ' ' )
        goto abort_3;
    if ( ( column[ 4 ] = getc( file ) ) != ' ' )
        goto abort_4;
    if ( isspace( column[ 5 ] = getc( file ) ) )
        goto abort_5;

    yylineno++;
    goto loop;

abort_5:
    if ( column[ 5 ] == '\t' )
        tab( 1 );
    else
    {
        yyunput( column[ 5 ] );
        if ( column[ 5 ] == '\n' )
```

```

        yylineno++;
    }
abort 4:
    if ( column[ 4 ] == '\t' )
        tab( 2 );
    else
    {
        yyunput( column[ 4 ] );
        if ( column[ 4 ] == '\n' )
            yylineno++;
    }
abort 3:
    if ( column[ 3 ] == '\t' )
        tab( 3 );
    else
    {
        yyunput( column[ 3 ] );
        if ( column[ 3 ] == '\n' )
            yylineno++;
    }
abort 2:
    if ( column[ 2 ] == '\t' )
        tab( 4 );
    else
    {
        yyunput( column[ 2 ] );
        if ( column[ 2 ] == '\n' )
            yylineno++;
    }
abort 1:
    if ( column[ 1 ] == '\t' )
        tab( 5 );
    else
    {
        yyunput( column[ 1 ] );
        if ( column[ 1 ] == '\n' )
            yylineno++;
    }
abort 0:
    if ( column[ 0 ] == '\t' )
        tab( 6 );
    else
    {
        yyunput( column[ 0 ] );
        if ( column[ 0 ] == '\n' )
            yylineno++;
    }
    return( c );
} /* yygetc */

```

FILE: network/communication/library/yywrap.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```

int yywrap( )
{
    return( 1 );
} /* yywrap */

```

FILE: network/communication/scanner.l

```

%{
/*
 * Copyright 1991
 * Georgia Institute of Technology

```

```

* Computer Engineering Research Laboratory
* Author: Stephen R. Wachtel
*/
%}

```

```

%a 10000
%e 10000
%k 10000
%n 10000
%o 10000
%p 10000

```

```

a  [aA]
b  [bB]
c  [cC]
d  [dD]
e  [eE]
f  [fF]
g  [gG]
h  [hH]
i  [iI]
j  [jJ]
k  [kK]
l  [lL]
m  [mM]
n  [nN]
o  [oO]
p  [pP]
q  [qQ]
r  [rR]
s  [sS]
t  [tT]
u  [uU]
v  [vV]
w  [wW]
x  [xX]
y  [yY]
z  [zZ]

```

```

%{
#include "grammar.h"
extern char *yylval;

```

```

#undef YYLMAX
#define YYLMAX (256*20)

```

```

extern char *duplicate( );
extern char *hollerith( );
extern char *non_blank( );
extern char *uppercase( );
%}

```

```

%%

```

```

^[\\*cC].*[\n]  |
^[\\ ]*[\n] {
#ifdef DEBUG
    ECHO;
#endif
    yyval = duplicate( yytext );
    return( COMMENT );
}

```

```

[\\ ] {
#ifdef DEBUG
    ECHO;
#endif
    /* return( '\\ ' ) */;
}

```

```

[\\&] {
#ifdef DEBUG

```

```
    ECHO;
#endif
    return( '\&' );
}

[ \ ( ] {
#ifdef DEBUG
    ECHO;
#endif
    return( '\(' );
}

[ \ ) ] {
#ifdef DEBUG
    ECHO;
#endif
    return( '\)' );
}

[ \ * ] {
#ifdef DEBUG
    ECHO;
#endif
    return( '\*' );
}

[ \ * ] [ \ * ] {
#ifdef DEBUG
    ECHO;
#endif
    return( EXPONENTIATE );
}

[ \ + ] {
#ifdef DEBUG
    ECHO;
#endif
    return( '\+' );
}

[ \ , ] {
#ifdef DEBUG
    ECHO;
#endif
    return( '\,' );
}

[ \ - ] {
#ifdef DEBUG
    ECHO;
#endif
    return( '\-' );
}

[ \ . ] {
#ifdef DEBUG
    ECHO;
#endif
    return( '\.' );
}

[ \ / ] {
#ifdef DEBUG
    ECHO;
#endif
    return( '\/' );
}

[ \ : ] {
#ifdef DEBUG
    ECHO;
#endif
```



```
#endif      return( '\:' );
}

[\\=] {
#ifdef DEBUG
    ECHO;
#endif
    return( '\\=' );
}

[\\n] {
#ifdef DEBUG
    ECHO;
#endif
    /* return( '\\n' ) */;
}

[\\t] {
#ifdef DEBUG
    ECHO;
#endif
    /* return( '\\t' ) */;
}

[\\.]{a}{n}{d}{\\.} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_AND );
}

[\\.]{e}{q}{\\.} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_EQ );
}

[\\.]{e}{q}{v}{\\.} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_EQV );
}

[\\.]{f}{a}{l}{s}{e}{\\.} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_FALSE );
}

[\\.]{g}{e}{\\.} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_GE );
}

[\\.]{g}{t}{\\.} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_GT );
}

[\\.]{l}{e}{\\.} {
#ifdef DEBUG
    ECHO;
#endif
}
```

```

        return( RW_LE );
    }

[\\.]{}{l}{t}{\\.} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_LT );
}

[\\.]{}{n}{e}{\\.} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_NE );
}

[\\.]{}{n}{e}{q}{v}{\\.} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_NEQV );
}

[\\.]{}{n}{o}{t}{\\.} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_NOT );
}

[\\.]{}{o}{r}{\\.} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_OR );
}

[\\.]{}{t}{r}{u}{e}{\\.} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_TRUE );
}

{a}{s}{s}{i}{g}{n} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_ASSIGN );
}

{b}{a}{c}{k}{s}{p}{a}{c}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_BACKSPACE );
}

{b}{l}{o}{c}{k}{[\\ ]*{d}{a}{t}{a}} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_BLOCK_DATA );
}

{c}{a}{l}{l} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_CALL );
}

```

```

    }

    {c}{h}{a}{r}{a}{c}{t}{e}{r} {
#ifdef DEBUG
        ECHO;
#endif
        return( RW_CHARACTER );
    }

    {c}{l}{o}{s}{e} {
#ifdef DEBUG
        ECHO;
#endif
        return( RW_CLOSE );
    }

    {c}{o}{m}{m}{o}{n} {
#ifdef DEBUG
        ECHO;
#endif
        return( RW_COMMON );
    }

    {c}{o}{m}{p}{l}{e}{x} {
#ifdef DEBUG
        ECHO;
#endif
        return( RW_COMPLEX );
    }

    {c}{o}{n}{t}{i}{n}{u}{e} {
#ifdef DEBUG
        ECHO;
#endif
        return( RW_CONTINUE );
    }

    {d}{a}{t}{a} {
#ifdef DEBUG
        ECHO;
#endif
        return( RW_DATA );
    }

    {d}{i}{m}{e}{n}{s}{i}{o}{n} {
#ifdef DEBUG
        ECHO;
#endif
        return( RW_DIMENSION );
    }

    {d}{o} {
#ifdef DEBUG
        ECHO;
#endif
        return( RW_DO );
    }

    {d}{o}{u}{b}{l}{e}[ \ ]*{p}{r}{e}{c}{i}{s}{i}{o}{n} {
#ifdef DEBUG
        ECHO;
#endif
        return( RW_DOUBLE_PRECISION );
    }

    {e}{l}{s}{e} {
#ifdef DEBUG
        ECHO;
#endif
        return( RW_ELSE );
    }

```

```

{e}{l}{s}{e}{\ }*{i}{f} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_ELSE_IF );
}

{e}{n}{d} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_END );
}

{e}{n}{d}{\ }*{i}{f} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_END_IF );
}

{e}{n}{d}{f}{i}{l}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_ENDFILE );
}

{e}{n}{t}{r}{y} {
#ifdef DEBUG
    ECHO;
#endif
    return( KW_ENTRY );
}

{e}{q}{u}{i}{v}{a}{l}{e}{n}{c}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_EQUIVALENCE );
}

{e}{x}{t}{e}{r}{n}{a}{l} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_EXTERNAL );
}

{f}{o}{r}{m}{a}{t}{t}{e}{r} {
#ifdef DEBUG
    ECHO;
#endif
    yyval = duplicate( yytext );
    return( RW_FORMAT );
}

{f}{u}{n}{c}{t}{i}{o}{n} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_FUNCTION );
}

{g}{o}{\ }*{t}{o} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_GO_TO );
}

```

```

{i}{f} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_IF );
}

{i}{m}{p}{l}{i}{c}{i}{t} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_IMPLICIT );
}

{i}{n}{c}{l}{u}{d}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_INCLUDE );
}

{i}{n}{q}{u}{i}{r}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_INQUIRE );
}

{i}{n}{t}{e}{g}{e}{r} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_INTEGER );
}

{i}{n}{t}{r}{i}{n}{s}{i}{c} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_INTRINSIC );
}

{l}{o}{g}{i}{c}{a}{l} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_LOGICAL );
}

{n}{a}{m}{e}{l}{i}{s}{t} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_NAMELIST );
}

{o}{p}{e}{n} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_OPEN );
}

{p}{a}{r}{a}{m}{e}{t}{e}{r} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_PARAMETER );
}

```

```

{p}{a}{u}{s}{e}    {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_PAUSE );
}

{p}{r}{i}{n}{t}    {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_PRINT );
}

{p}{r}{o}{g}{r}{a}{m} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_PROGRAM );
}

{r}{e}{a}{d}    {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_READ );
}

{r}{e}{a}{l}    {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_REAL );
}

{r}{e}{t}{u}{r}{n} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_RETURN );
}

{r}{e}{w}{i}{n}{d} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_REWIND );
}

{s}{a}{v}{e}    {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_SAVE );
}

{s}{t}{o}{p}    {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_STOP );
}

{s}{u}{b}{r}{o}{u}{t}{i}{n}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_SUBROUTINE );
}

```

```

{t}{h}{e}{n} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_THEN );
}

{t}{o} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_TO );
}

{w}{r}{i}{t}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_WRITE );
}

{u}{n}{d}{e}{f}{i}{n}{e}{d} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_UNDEFINED );
}

[%a-zA-Z][_a-zA-Z0-9]* {
#ifdef DEBUG
    ECHO;
#endif
    yylval = duplicate( uppercase( yytext ) );
    return( IDENTIFIER );
}

^[0-9][0-9][0-9][0-9][0-9] \ {
#ifdef DEBUG
    ECHO;
#endif
    yylval = duplicate( non_blank( yytext ) );
    return( LABEL );
}

[0-9]+ |
[0-9]+/\.[a-zA-Z]+\.\ {
#ifdef DEBUG
    ECHO;
#endif
    yylval = duplicate( yytext );
    return( INTEGER );
}

[0-9]+\.[0-9]*([eE](\+|-)?[0-9]+)? |
[0-9]*\.[0-9]+([eE](\+|-)?[0-9]+)? |
[0-9]+([eE](\+|-)?[0-9]+)? {
#ifdef DEBUG
    ECHO;
#endif
    yylval = duplicate( yytext );
    return( REAL );
}

[0-9]+\.[0-9]*([dD](\+|-)?[0-9]+)? |
[0-9]*\.[0-9]+([dD](\+|-)?[0-9]+)? |
[0-9]+([dD](\+|-)?[0-9]+)? {
#ifdef DEBUG
    ECHO;
#endif
    yylval = duplicate( yytext );
    return( DOUBLE_PRECISION );
}

```

```

\[^\']*\\' |
\[^\"]*\\ " {
#ifdef DEBUG
    ECHO;
#endif
    yytext[ 0 ] = '\\';
    yytext[ strlen( yytext ) - 1 ] = '\\';
    yylval = duplicate( yytext );
    return( STRING );
}

[0-9]+[hH] {
#ifdef DEBUG
    ECHO;
#endif
    yylval = duplicate( hollerith( yytext, '\\ ' ) );
    return( HOLLERITH );
}

```

FILE: network/network/Makefile

```

#
# Copyright 1991
# Georgia Institute of Technology
# Computer Engineering Research Laboratory
# Author: Stephen R. Wachtel
#

default:    network

CC = cc -g
INCLUDE = include
CFLAGS = -I$(INCLUDE)
LIBRARY = library/library.a

network.o: network.c
    $(CC) $(CFLAGS) -c network.c

network:    network.o $(LIBRARY)
    $(CC) $(CFLAGS) -o network network.o $(LIBRARY)

clean:
    cd library; make clean
    rm -f network network.o

```

FILE: network/network/include/list.h

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#define LIST struct list_type
LIST
{
    char *identifier;
    char message_type;
    char usage;
    int priority;
    LIST *next;
};

extern LIST *end_list( );
extern LIST *add_list( );
extern LIST *find_list( );
extern void print_list( );

```



```
extern void delete_list( );
```

```
FILE: network/network/include/processor.h
```

```
/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */
```

```
#define NUMBER_PROCESSOR 32
```

```
#define PROCESSOR struct processor_type
PROCESSOR
```

```
{
    char *file_name;
    int s;
    int r;
    LIST *list;

    char usage[ NUMBER_PROCESSOR + 1 ];
};
```

```
extern PROCESSOR processor[ NUMBER_PROCESSOR ];
```

```
FILE: network/network/library/Makefile
```

```
#
# Copyright 1991
# Georgia Institute of Technology
# Computer Engineering Research Laboratory
# Author: Stephen R. Wachtel
#
```

```
CC = cc -g
INCLUDE = ../include
CFLAGS = -I$(INCLUDE)
LIBRARY = library.a
```

```
OBJECTS = \
    count.o \
    duplicate.o \
    link_list.o \
    message_type_length.o \
    message_type_name.o
```

```
$(LIBRARY): $(OBJECTS)
    ar crv $(LIBRARY) $(OBJECTS)
    ranlib $(LIBRARY)
```

```
.SUFFIXES: .c .o
.c.o:
    $(CC) -c $(CFLAGS) $<
```

```
clean:
    rm -f $(LIBRARY) $(OBJECTS)
```

```
FILE: network/network/library/count.c
```

```
/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */
```

```

int count( string, length, c )
register char *string;
register int length;
register char c;
{
    register int c_count = 0;

    while ( length != 0 )
    {
        if ( *string == c )
            c_count++;

        string++;
        length--;
    }

    return( c_count );
} /* count */

```

FILE: network/network/library/duplicate.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include <stdio.h>
#include <string.h>
#include <malloc.h>

char *duplicate( string )
register char *string;
{
    register char *temporary = (char *)NULL;

    if ( string != (char *)NULL )
    {
        if ( ( temporary = (char *)malloc( strlen( string ) + 1 ) ) != (char *)NULL )
            strcpy( temporary, string );
        else
            fprintf( stderr, "ERROR: duplicate( %s )\n", string );
    }

    return( temporary );
} /* duplicate */

```

FILE: network/network/library/link\_list.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include <stdio.h>
#include <string.h>
#include <malloc.h>
#include "list.h"

extern char *duplicate( );

LIST *end_list( list )
register LIST *list;
{
    if ( list != (LIST *)NULL )
    {
        while ( list->next != (LIST *)NULL )
            list = list->next;
    }
}

```

```

    }

    return( list );
} /* end_list */

LIST *add_list( list, identifier, message_type, usage, priority )
register LIST **list;
register char *identifier;
register char *message_type;
register char usage;
register int priority;
{
    register LIST *temporary = (LIST *)malloc( sizeof( LIST ) );

    temporary->identifier = duplicate( identifier );
    temporary->message_type = duplicate( message_type );
    temporary->usage = usage;
    temporary->priority = priority;
    temporary->next = (LIST *)NULL;

    if ( *list == (LIST *)NULL )
        *list = temporary;
    else
        end_list( *list )->next = temporary;

    return( temporary );
} /* add_list */

LIST *find_list( list, identifier, message_type )
register LIST *list;
register char *identifier;
register char *message_type;
{
    while ( list != (LIST *)NULL )
    {
        if ( ( strcmp( list->identifier, identifier ) == 0 )
            && ( strcmp( list->message_type, message_type ) == 0 ) )
            return( list );

        list = list->next;
    }

    return( (LIST *)NULL );
} /* find_list */

void print_list( file, list )
register FILE *file;
register LIST *list;
{
    while ( list != (LIST *)NULL )
    {
        switch ( list->usage )
        {
            case 'S':
                fprintf( file, "\tCALL SEND_%s( %s )\n", list->message_type, list->
                identifier );
                break;

            case 'R':
                fprintf( file, "\tCALL RECEIVE_%s( %s )\n", list->message_type, list->
                identifier );
                break;
        }

        list = list->next;
    }
} /* print_list */

void delete_list( list, identifier, message_type )
register LIST **list;
register char *identifier;
register char *message_type;
{
    register LIST *last = (LIST *)NULL;
    register LIST *curr = *list;

    while ( curr != (LIST *)NULL )

```

```

    {
        if ( ( strcmp( curr->identifier, identifier ) == 0 )
            && ( strcmp( curr->message_type, message_type ) == 0 ) )
        {
            if ( last == (LIST *)NULL )
                *list = curr->next;
            else
                last->next = curr->next;

            break;
        }

        last = curr;
        curr = curr->next;
    }
} /* delete_list */

```

FILE: network/network/library/message\_type\_length.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include <stdio.h>
#include <string.h>

int message_type_length( message_type )
register char *message_type;
{
    if ( strcmp( "CHARACTER_08BIT", message_type ) == 0 )
        return( 2 );

    if ( strcmp( "COMPLEX_32BIT", message_type ) == 0 )
        return( 8 );
    if ( strcmp( "COMPLEX_64BIT", message_type ) == 0 )
        return( 16 );

    if ( strcmp( "LOGICAL_08BIT", message_type ) == 0 )
        return( 2 );
    if ( strcmp( "LOGICAL_16BIT", message_type ) == 0 )
        return( 2 );
    if ( strcmp( "LOGICAL_32BIT", message_type ) == 0 )
        return( 4 );

    if ( strcmp( "REAL_32BIT", message_type ) == 0 )
        return( 4 );
    if ( strcmp( "REAL_64BIT", message_type ) == 0 )
        return( 8 );

    if ( strcmp( "SIGNED_08BIT", message_type ) == 0 )
        return( 2 );
    if ( strcmp( "SIGNED_16BIT", message_type ) == 0 )
        return( 2 );
    if ( strcmp( "SIGNED_32BIT", message_type ) == 0 )
        return( 4 );

    if ( strcmp( "UNSIGNED_08BIT", message_type ) == 0 )
        return( 2 );
    if ( strcmp( "UNSIGNED_16BIT", message_type ) == 0 )
        return( 2 );
    if ( strcmp( "UNSIGNED_32BIT", message_type ) == 0 )
        return( 4 );

    fprintf( stderr, "ERROR: message_type_length( %s )\n", message_type );
    exit( -1 );
} /* message_type_length */

```

FILE: network/network/library/message\_type\_name.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology

```

```

* Computer Engineering Research Laboratory
* Author: Stephen R. Wachtel
*/

```

```

#include <stdio.h>
#include <string.h>

```

```

char *message_type_name( message_type )
register char *message_type;
{
    if ( strcmp( "CHARACTER_08BIT", message_type ) == 0 )
        return( "CHARACTER*1" );

    if ( strcmp( "COMPLEX_32BIT", message_type ) == 0 )
        return( "COMPLEX*8" );
    if ( strcmp( "COMPLEX_64BIT", message_type ) == 0 )
        return( "COMPLEX*16" );

    if ( strcmp( "LOGICAL_08BIT", message_type ) == 0 )
        return( "LOGICAL*1" );
    if ( strcmp( "LOGICAL_16BIT", message_type ) == 0 )
        return( "LOGICAL*2" );
    if ( strcmp( "LOGICAL_32BIT", message_type ) == 0 )
        return( "LOGICAL*4" );

    if ( strcmp( "REAL_32BIT", message_type ) == 0 )
        return( "REAL*4" );
    if ( strcmp( "REAL_64BIT", message_type ) == 0 )
        return( "REAL*8" );

    if ( strcmp( "SIGNED_08BIT", message_type ) == 0 )
        return( "INTEGER*1" );
    if ( strcmp( "SIGNED_16BIT", message_type ) == 0 )
        return( "INTEGER*2" );
    if ( strcmp( "SIGNED_32BIT", message_type ) == 0 )
        return( "INTEGER*4" );

    if ( strcmp( "UNSIGNED_08BIT", message_type ) == 0 )
        return( "UNSIGNED_INTEGER*1" );
    if ( strcmp( "UNSIGNED_16BIT", message_type ) == 0 )
        return( "UNSIGNED_INTEGER*2" );
    if ( strcmp( "UNSIGNED_32BIT", message_type ) == 0 )
        return( "UNSIGNED_INTEGER*4" );

    fprintf( stderr, "ERROR: message_type_name( %s )\n", message_type );
    exit( -1 );
} /* message_type_name */

```

```

FILE: network/network/network.c

```

```

/*
* Copyright 1991
* Georgia Institute of Technology
* Computer Engineering Research Laboratory
* Author: Stephen R. Wachtel
*/

```

```

#include <stdio.h>
#include <string.h>
#include "list.h"
#include "processor.h"

```

```

extern char *duplicate( );
extern char *message_type_name( );

```

```

PROCESSOR processor[ NUMBER_PROCESSOR ];

```

```

#define PRIORITY 1000
#define DEFAULT_PRIORITY ( PRIORITY * PRIORITY )

```

```

LIST *priority_list = (LIST *)NULL;
#define MULTIPLE_TRANSFER 1

```

```

void input_priority_list( file )
register FILE *file;
{
    char line[ 256 ];
    int priority = PRIORITY;

    while ( fgets( line, sizeof( line ), file ) != (char *)NULL )
    {
        line[ strlen( line ) - 1 ] = '\0';

        if ( line[ 0 ] == '#' )
        {
            priority = PRIORITY + ( ( priority / PRIORITY ) * PRIORITY );
            continue;
        }

        add_list( &priority_list, duplicate( line ), 0, '\0', priority++ );
    }
} /* input_priority_list */

int find_priority_list( identifier )
register char *identifier;
{
    char temporary[ 256 ];
    register LIST *list;

    strcpy( temporary, identifier );
    temporary[ strcspn( temporary, "(" ) ] = '\0';

    list = priority_list;
    while ( list != (LIST *)NULL )
    {
        if ( strcmp( list->identifier, temporary ) == 0 )
            return( list->priority );

        list = list->next;
    }

    return( DEFAULT_PRIORITY );
} /* find_priority_list */

void initialize_processor( )
{
    register int processor_number;

    for ( processor_number = 0; processor_number != NUMBER_PROCESSOR; processor_number++ )
    {
        processor[ processor_number ].file_name = (char *)NULL;
        processor[ processor_number ].s = 0;
        processor[ processor_number ].r = 0;
        processor[ processor_number ].list = (LIST *)NULL;
    }
} /* initialize_processor */

void input_list( processor_number )
register int processor_number;
{
    FILE *file;
    char line[ 256 ];
    char identifier[ 256 ];
    char message_type[ 256 ];
    char usage;
    int priority;

    if ( ( file = fopen( processor[ processor_number ].file_name, "r" ) ) == (FILE *)NULL )
    {
        fprintf( stderr, "ERROR: unable to open for read '%s'\n", processor[
processor_number ].file_name );
        exit( -1 );
    }

    while ( fgets( line, sizeof( line ), file ) != (char *)NULL )
    {
        if ( sscanf( line, "%s %s %c\n", identifier, message_type, &usage ) != 3 )
            continue;
    }
}

```

```

    fprintf( stderr, "ERROR: unable to parse line '%s'\n", line );
    exit( -1 );
}

if ( strcmp( message_type, "REAL_32BIT_AS_64BIT" ) == 0 )
    strcpy( message_type, "REAL_32BIT" );

if ( strcmp( message_type, "REAL_64BIT_AS_32BIT" ) == 0 )
    strcpy( message_type, "REAL_64BIT" );

switch ( usage )
{
    case 'S':
        processor[ processor_number ].s++;
        priority = find_priority_list( identifier );
        break;

    case 'R':
        processor[ processor_number ].r++;
        priority = DEFAULT_PRIORITY;
        break;
}

add_list( &processor[ processor_number ].list, identifier, message_type, usage,
priority );
}

fclose( file );
} /* input_list */

void output_transfer( file, identifier, message_type, usage )
register FILE *file;
register char *identifier;
register char *message_type;
register char *usage;
{
    register char delimiter;
    register int s, r;
    register int priority;

    delimiter = ' ';
    for ( r = 0; r != NUMBER_PROCESSOR; r++ )
    {
        if ( usage[ r ] == 'R' )
        {
            priority = processor[ r ].list->priority;

            fprintf( file, "%c p%02d", delimiter, r );
            delete_list( &processor[ r ].list, identifier, message_type );

            delimiter = ',';
        }
    }

    fprintf( file, " := " );

    delimiter = ' ';
    for ( s = 0; s != NUMBER_PROCESSOR; s++ )
    {
        if ( usage[ s ] == 'S' )
        {
            priority = processor[ s ].list->priority;

            fprintf( file, "%c p%02d", delimiter, s );
            delete_list( &processor[ s ].list, identifier, message_type );

            delimiter = ',';
        }
    }

    fprintf( file, ".%d; { %s %s %d }\n", message_type_length( message_type ) / 2,
message_type_name( message_type ), identifier, priority );
} /* output_transfer */

int and_usage( usage1, usage2 )
register char *usage1;
register char *usage2;
{
    register int processor_number = 0;

```

```

    while ( usage1[ processor_number ] != '\0' ) && ( usage2[ processor_number ] != '\0'
) )
    {
        if ( ( usage1[ processor_number ] != '-' ) && ( usage2[ processor_number ] != '-'
) )
            return( 1 );

        processor_number++;
    }

    return( 0 );
} /* and_usage */

char *or_usage( usage1, usage2 )
register char *usage1;
register char *usage2;
{
    static char usage[ NUMBER_PROCESSOR + 1 ];
    register int processor_number = 0;

    while ( ( usage1[ processor_number ] != '\0' ) && ( usage2[ processor_number ] != '\0'
) )
    {
        usage[ processor_number ] = '-';

        if ( usage1[ processor_number ] == ' ' )
            usage[ processor_number ] = usage2[ processor_number ];

        if ( usage2[ processor_number ] == '-' )
            usage[ processor_number ] = usage1[ processor_number ];

        processor_number++;
    }
    usage[ processor_number ] = '\0';

    return( usage );
} /* or_usage */

int usage_cycle( )
{
    register int s, r;
    register LIST *list;
    register int number_transfer = 0;

    for ( s = 0; s != NUMBER_PROCESSOR; s++ )
    {
        strcpy( processor[ s ].usage, "-----" );

        if ( processor[ s ].list == (LIST *)NULL )
            continue;

        if ( processor[ s ].list->usage != 'S' )
            continue;

        processor[ s ].usage[ s ] = 'S';

        for ( r = 0; r != NUMBER_PROCESSOR; r++ )
        {
            if ( processor[ r ].list == (LIST *)NULL )
                continue;

            if ( r == s )
                continue;

            if ( ( list = find_list( processor[ r ].list, processor[ s ].list->identifier,
processor[ s ].list->message_type ) ) != (LIST *)NULL )
            {
                if ( list != processor[ r ].list )
                {
                    strcpy( processor[ s ].usage, "-----" );

                    goto incomplete;
                }

                processor[ s ].usage[ r ] = 'R';
            }
        }
    }
}

```



```

        number_transfer++;

incomplete:
    ;

    return( number_transfer );
} /* usage_cycle */

int next_sender( )
{
    register int s = 0;
    register int r;
    register int priority;

    s = NUMBER_PROCESSOR;
    priority = DEFAULT_PRIORITY + 1;

    for ( r = 0; r != NUMBER_PROCESSOR; r++ )
    {
        if ( processor[ r ].list == (LIST *)NULL )
            continue;

        if ( count( processor[ r ].usage, strlen( processor[ r ].usage ), '-' ) !=
NUMBER_PROCESSOR )
        {
            if ( processor[ r ].list->priority < priority )
            {
                s = r;
                priority = processor[ r ].list->priority;
            }
        }
    }

    return( s );
} /* next_sender */

void output_cycle( file, usage )
register FILE *file;
register char *usage;
{
    register int s, r;
    register int priority = DEFAULT_PRIORITY + 1;

    while ( ( s = next_sender( ) ) != NUMBER_PROCESSOR )
    {
        if ( and_usage( usage, processor[ s ].usage ) == 0 )
        {
            if ( processor[ s ].list->priority < priority )
            {
                priority = processor[ s ].list->priority / PRIORITY *
PRIORITY );
                strcpy( usage, or_usage( usage, processor[ s ].usage ) );
                output_transfer( file, processor[ s ].list->identifier, processor[ s
].list->message_type, processor[ s ].usage );
            }
        }

        #ifndef MULTIPLE_TRANSFER
            break;
        #endif
    }

    strcpy( processor[ s ].usage, "-----" );
} /* output_cycle */

void output_network( file )
register FILE *file;
{
    char usage[ NUMBER_PROCESSOR + 1 ];
    register int cycle = 0;

    fprintf( file, "LOOP\n\n" );

    while ( usage_cycle( ) != 0 )
    {
        strcpy( usage, "-----" );
    }
}

```

```

        fprintf( file, "CYCLE [ %d ]\n", ++cycle );
        output_cycle( file, usage );
        fprintf( file, "\n" );
    }
} /* output_network */

void output_processor( file )
register FILE *file;
{
    register int number;
    register int number_error = 0;

    for ( number = 0; number != NUMBER_PROCESSOR; number++ )
    {
        if ( processor[ number ].file_name != (char *)NULL )
        {
            fprintf( file, "[ " );

            strcpy( index( processor[ number ].file_name, '.' ), ".for" );
            fprintf( file, "p%02d = %s", number, processor[ number ].file_name );

            fprintf( file, ", S = %3d", processor[ number ].s );
            fprintf( file, ", R = %3d", processor[ number ].r );
            fprintf( file, ", %3d", processor[ number ].s + processor[ number ].r );

            fprintf( file, " ]\n" );

            if ( processor[ number ].list != (LIST *)NULL )
            {
                print_list( file, processor[ number ].list );
                fprintf( file, "\n" );

                number_error++;
            }
        }
    }

    if ( number_error != 0 )
    {
        fprintf( file, "ERROR: %d processor(s) with unresolved communication\n",
            number_error );
        fprintf( stderr, "ERROR: %d processor(s) with unresolved communication\n",
            number_error );
    }
} /* output_processor */

#define PROGRAM argument[ 0 ]
#define ARGUMENT argument[ argument_number ]

int main( number_argument, argument )
register int number_argument;
register char *argument[ ];
{
    register int argument_number = 0;
    int processor_number;
    char file_name[ 256 ];

    if ( --number_argument == 0 )
    {
        fprintf( stderr, "usage: %s 00=<file name>...31=<file name>\n", PROGRAM );
        exit( 0 );
    }

    input_priority_list( stdin );
    initialize_processor( );

    while ( argument_number++ != number_argument )
    {
        if ( sscanf( ARGUMENT, "%d=%s", &processor_number, file_name ) != 2 )
        {
            fprintf( stderr, "ERROR: unable to parse argument '%s'\n", ARGUMENT );
            exit( -1 );
        }

        processor[ processor_number ].file_name = duplicate( file_name );
        input_list( processor_number );
    }
}

```

```
output_network( stdout );  
output_processor( stdout );  
  
exit( 0 );  
} /* main */
```

**17. Appendix L: structure program source**

FILE: structure/Makefile

```
#
# Copyright 1991
# Georgia Institute of Technology
# Computer Engineering Research Laboratory
# Author: Stephen R. Wachtel
#

default:    structure

CC = cc -g
INCLUDE = include
CFLAGS = -I$(INCLUDE)
LIBRARY = library/library.a

OBJECTS = \
    $(INCLUDE)/grammar.h \
    *grammar.[co] \
    *scanner.[co] \
    yytrace.[co] \
    y.output

PROGRAMS = \
    *structure

grammar.c: grammar.y
    yacc -dv grammar.y
    mv y.tab.h $(INCLUDE)/grammar.h
    mv y.tab.c grammar.c

scanner.c: scanner.l
    lex -vt scanner.l | sed 's/getc/yygetc/' >scanner.c

scanner.o: scanner.c $(INCLUDE)/grammar.h
    $(CC) $(CFLAGS) -c scanner.c

grammar.o: grammar.c
    $(CC) $(CFLAGS) -c grammar.c

structure: grammar.o scanner.o $(LIBRARY)
    $(CC) -o structure grammar.o scanner.o $(LIBRARY)

sgrammar.c: grammar.c yytoken.awk
    awk -f yytoken.awk <grammar.c >sgrammar.c

sgrammar.o: sgrammar.c
    $(CC) $(CFLAGS) -c sgrammar.c

sstructure: sgrammar.o scanner.o $(LIBRARY)
    $(CC) -o sstructure sgrammar.o scanner.o $(LIBRARY)

dscanner.c: scanner.c
    cp scanner.c dscanner.c

dscanner.o: dscanner.c $(INCLUDE)/grammar.h
    $(CC) $(CFLAGS) -DDEBUG -c dscanner.c

dstructure: grammar.o dscanner.o $(LIBRARY)
    $(CC) -o dstructure grammar.o dscanner.o $(LIBRARY)

tgrammar.c: grammar.c
    sed 's/yystate/& yytrace(yystate);/' <grammar.c >tgrammar.c

tgrammar.o: tgrammar.c
    $(CC) $(CFLAGS) -c tgrammar.c
```

```
tstructure: tgrammar.o scanner.o yytrace.o $(LIBRARY)
$(CC) -o tstructure tgrammar.o scanner.o yytrace.o $(LIBRARY)
```

```
yytrace.c: grammar.c yytrace.awk
awk -f yytrace.awk <y.output >yytrace.c
```

```
yytrace.o: yytrace.c
$(CC) $(CFLAGS) -c yytrace.c
```

```
clean:
cd library; make clean
rm -f $(PROGRAMS) $(OBJECTS)
```

```
FILE: structure/grammar.y
```

```
/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */
```

```
/*
 * FORTRAN 77
 */
```

```
%token RW_AND
%token RW_ASSIGN
%token RW_BACKSPACE
%token RW_BLOCK_DATA
%token RW_CALL
%token RW_CHARACTER
%token RW_CLOSE
%token RW_COMMON
%token RW_COMPLEX
%token RW_CONTINUE
%token RW_DATA
%token RW_DIMENSION
%token RW_DO
%token RW_DOUBLE_PRECISION
%token RW_ELSE
%token RW_ELSE_IF
%token RW_END
%token RW_END_IF
%token RW_ENDFILE
%token RW_ENTRY
%token RW_EQ
%token RW_EQUIVALENCE
%token RW_EQV
%token RW_EXTERNAL
%token RW_FALSE
%token RW_FORMAT
%token RW_FUNCTION
%token RW_GE
%token RW_GO_TO
%token RW_GT
%token RW_IF
%token RW_IMPLICIT
%token RW_INCLUDE
%token RW_INQUIRE
%token RW_INTEGER
%token RW_INTRINSIC
%token RW_LE
%token RW_LOGICAL
%token RW_LT
%token RW_NAMELIST
%token RW_NE
%token RW_NEQV
%token RW_NOT
%token RW_OPEN
%token RW_OR
%token RW_PARAMETER
%token RW_PAUSE
%token RW_PRINT
%token RW_PROGRAM
```

```

%token RW_READ
%token RW_REAL
%token RW_RETURN
%token RW_REWIND
%token RW_SAVE
%token RW_STOP
%token RW_SUBROUTINE
%token RW_THEN
%token RW_TO
%token RW_TRUE
%token RW_WRITE
%token RW_UNDEFINED

%token COMMENT
%token CONCATENATE
%token DOUBLE_PRECISION
%token EXPONENTIATE
%token HOLLERITH
%token IDENTIFIER
%token INTEGER
%token LABEL
%token REAL
%token STRING

%left ','
%nonassoc ':'
%right '='
%left RW_EQV RW_NEQV
%left RW_OR
%left RW_AND
%left RW_NOT
%nonassoc RW_EQ RW_NE RW_LT RW_LE RW_GT RW_GE
%left CONCATENATE
%left '+' '-'
%left '*' '/'
%right EXPONENTIATE
%left SIGN

%{
typedef char *POINTER;
#define YYSTYPE POINTER

#include "list.h"
static LIST *block_list = 0;
static LIST *subprogram_list;
static LIST *array_list;

static POINTER block_name;
static int block_type;
%}

%%

program:
    optional_statement_list
    {
        summary( block_list );
    }
    ;

optional_statement_list:
    /* NULL */
    |
        statement_list
    ;

statement_list:
    statement
    |
        statement_list statement
    ;

statement:

```

```

    comment_statement
  |
  label unlabeled_statement
;

comment_statement:
  COMMENT
;

label:
  LABEL
;

unlabeled_statement:
  include_statement
  |
  program_statement
  |
  block_data_statement
  |
  function_statement
  |
  subroutine_statement
  |
  entry_statement
  |
  end_statement
  |
  specification_statement
  |
  executable_statement
  |
  format_statement
;

include_statement:
  RW_INCLUDE character_constant
;

program_statement:
  RW_PROGRAM program_identifier
;

program_identifier:
  IDENTIFIER
  {
    block_name = $1;
    block_type = RW_PROGRAM;
    array_list = 0;
    subprogram_list = 0;
  }
;

block_data_statement:
  RW_BLOCK_DATA block_data_identifier
;

block_data_identifier:
  IDENTIFIER
  {
    block_name = $1;
    block_type = RW_BLOCK_DATA;
    array_list = 0;
    subprogram_list = 0;
  }
;

function_statement:
  RW_FUNCTION function_identifier optional_formal_argument_list
  |
  type RW_FUNCTION function_identifier optional_formal_argument_list
;

```

```

function_identifier:
    IDENTIFIER
    {
        block_name = $1;
        block_type = RW_FUNCTION;
        array_list = 0;
        subprogram_list = 0;
    }
;

subroutine_statement:
    RW_SUBROUTINE subroutine_identifier
    |
    RW_SUBROUTINE subroutine_identifier optional_formal_argument_list
;

subroutine_identifier:
    IDENTIFIER
    {
        block_name = $1;
        block_type = RW_SUBROUTINE;
        array_list = 0;
        subprogram_list = 0;
    }
;

entry_statement:
    RW_ENTRY entry_identifier
    |
    RW_ENTRY entry_identifier optional_formal_argument_list
;

entry_identifier:
    IDENTIFIER
;

optional_formal_argument_list:
    '(' ' ' ')'
    |
    '(' formal_argument_list ')'
;

formal_argument_list:
    formal_argument
    |
    formal_argument_list ',' formal_argument
;

formal_argument:
    IDENTIFIER
    |
    formal_argument_alterate_return
;

formal_argument_alterate_return:
    '*'
;

end_statement:
    RW_END
    {
        add_list( &block_list, block_name, block_type, subprogram_list );
        delete_list( array_list );
    }
;

specification_statement:
    external_statement

```



```

|
|   intrinsic_statement
|
|   parameter_statement
|
|   dimension_statement
|
|   declaration_statement
|
|   save_statement
|
|   common_statement
|
|   equivalence_statement
|
|   implicit_statement
|
|   data_statement
|
|   namelist_statement
;

external_statement:
    RW_EXTERNAL external_list
;

external_list:
    external
|
    external_list ',' external
;

external:
    IDENTIFIER
;

intrinsic_statement:
    RW_INTRINSIC intrinsic_list
;

intrinsic_list:
    intrinsic
|
    intrinsic_list ',' intrinsic
;

intrinsic:
    IDENTIFIER
;

parameter_statement:
    RW_PARAMETER '(' parameter_list ')'
;

parameter_list:
    parameter
|
    parameter_list ',' parameter
;

parameter:
    IDENTIFIER '=' expression
;

dimension_statement:
    RW_DIMENSION dimension_list
;

dimension_list:
    dimension

```

```

|
| dimension_list ',' dimension
;

dimension:
    IDENTIFIER '(' subscript_list ')'
    {
        if ( find_list( array_list, $1 ) == 0 )
        {
            add_list( &array_list, $1, 0, 0 );
        }
    }
;

subscript_list:
    subscript
|
| subscript_list ',' subscript
;

subscript:
    upper_bound
|
| lower_bound ':' upper_bound
;

lower_bound:
    expression
;

upper_bound:
    lower_bound
|
| upper_bound_adjustable
;

upper_bound_adjustable:
    '*'
;

declaration_statement:
    type declaration_list
;

declaration_list:
    declaration
|
| declaration_list ',' declaration
;

declaration:
    IDENTIFIER optional_type_length
|
| IDENTIFIER '(' subscript_list ')' optional_type_length
    {
        if ( find_list( array_list, $1 ) == 0 )
        {
            add_list( &array_list, $1, 0, 0 );
        }
    }
;

type:
    type_name optional_type_length
;

type_name:
    RW_CHARACTER
|
| RW_COMPLEX

```

```

    |
    | RW_DOUBLE_PRECISION
    |
    | RW_INTEGER
    |
    | RW_LOGICAL
    |
    | RW_REAL
    |
    | RW_UNDEFINED
    ;

optional_type_length:
    /* NULL */
    |
    | type_length
    ;

type_length:
    '*' INTEGER
    |
    | '*' type_length_adjustable
    ;

type_length_adjustable:
    '(' '*' ')'
    ;

save_statement:
    RW_SAVE optional_save_list
    ;

optional_save_list:
    /* NULL */
    |
    | save_list
    ;

save_list:
    save
    |
    | save_list ',' save
    ;

save:
    IDENTIFIER
    |
    | common_name
    ;

common_statement:
    RW_COMMON optional_common_name common_list
    ;

optional_common_name:
    /* NULL */
    |
    | common_name
    ;

common_name:
    '/' optional_identifier '/'
    ;

optional_identifier:
    /* NULL */
    |
    | IDENTIFIER
    ;

```

```

common_list:
  common
  |
  |   common_list ',' common
  ;

common:
  IDENTIFIER
  |
  |   IDENTIFIER '(' subscript_list ')'
  |   {
  |       if ( find_list( array_list, $1 ) == 0 )
  |       {
  |           add_list( &array_list, $1, 0, 0 );
  |       }
  |   }
  ;

equivalence_statement:
  RW_EQUIVALENCE equivalence_list
  ;

equivalence_list:
  equivalence
  |
  |   equivalence_list ',' equivalence
  ;

equivalence:
  '(' variable_list ')'
  ;

variable_list:
  variable
  |
  |   variable_list ',' variable
  ;

implicit_statement:
  RW_IMPLICIT type '(' implicit_list ')'
  ;

implicit_list:
  implicit
  |
  |   implicit_list ',' implicit
  ;

implicit:
  IDENTIFIER
  |
  |   IDENTIFIER '-' IDENTIFIER
  ;

namelist_statement:
  RW_NAMELIST namelist_name namelist_list
  ;

namelist_name:
  '/' IDENTIFIER '/'
  ;

namelist_list:
  namelist
  |
  |   namelist_list ',' namelist
  ;

```

```

namelist:
    IDENTIFIER
    ;

data_statement:
    RW_DATA data_list
    ;

data_list:
    data
    |
    data_list optional_comma data
    ;

data:
    data_variable_list '/' data_constant_list '/'
    ;

data_variable_list:
    data_variable
    |
    data_variable_list ',' data_variable
    ;

data_variable:
    variable
    |
    data_implied_do_list
    ;

data_implied_do_list:
    '(' data_variable_list ',' IDENTIFIER '=' expression_list ')'
    ;

data_constant_list:
    data_constant
    |
    data_constant_list ',' data_constant
    ;

data_constant:
    data_initialization
    |
    IDENTIFIER '*' data_initialization
    |
    INTEGER '*' data_initialization
    ;

data_initialization:
    IDENTIFIER
    |
    character_constant
    |
    logical_constant
    |
    signed_numerical_constant
    ;

signed_numerical_constant:
    numerical_constant
    |
    '+' numerical_constant %prec SIGN
    |
    '-' numerical_constant %prec SIGN
    ;

expression:
    parenthesis_expression
    |
    simple_expression

```

```

;

parenthesis_expression:
    '(' expression ')'
;

simple_expression:
    variable
    |
    constant
    |
    arithmetic_expression
    |
    character_expression
    |
    relational_expression
    |
    logical_expression
    |
    unary_expression
;

variable:
    IDENTIFIER
    |
    IDENTIFIER string_subset
    |
    array
;

array:
    IDENTIFIER '(' optional_expression_list ')'
    {
        if ( find_list( array_list, $1 ) == 0 )
        {
            add_list( &subprogram_list, $1, 0, 0 );
        }
    }
    |
    IDENTIFIER '(' optional_expression_list ')' string_subset
    {
        if ( find_list( array_list, $1 ) == 0 )
        {
            add_list( &subprogram_list, $1, 0, 0 );
        }
    }
;

optional_expression_list:
    /* NULL */
    |
    expression_list
;

expression_list:
    expression
    |
    expression_list ',' expression
;

string_subset:
    '(' optional_expression ':' optional_expression ')'
;

optional_expression:
    /* NULL */
    |
    expression
;

constant:
    character_constant

```

```

    logical_constant
    numerical_constant
;

character_constant:
    HOLLERITH
    STRING
;

logical_constant:
    RW_FALSE
    RW_TRUE
;

numerical_constant:
    DOUBLE_PRECISION
    INTEGER
    REAL
;

arithmetic_expression:
    expression '+' expression %prec '+'
    expression '-' expression %prec '-'
    expression '*' expression %prec '*'
    expression '/' expression %prec '/'
    expression EXPONENTIATE expression %prec EXPONENTIATE
;

character_expression:
    expression '/' '/' expression %prec CONCATENATE
;

relational_expression:
    expression RW_EQ expression %prec RW_EQ
    expression RW_NE expression %prec RW_NE
    expression RW_LT expression %prec RW_LT
    expression RW_LE expression %prec RW_LE
    expression RW_GT expression %prec RW_GT
    expression RW_GE expression %prec RW_GE
;

logical_expression:
    expression RW_AND expression %prec RW_AND
    expression RW_OR expression %prec RW_OR
    expression RW_EQV expression %prec RW_EQV
    expression RW_NEQV expression %prec RW_NEQV
;

unary_expression:
    '-' expression %prec SIGN
    '-' expression %prec SIGN
    RW_NOT expression %prec RW_NOT
;

```

```

executable_statement:
    do_statement
    |
    logical_if_statement
    |
    block_if_statement
    |
    else_statement
    |
    else_if_statement
    |
    end_if_statement
    |
    subset_executable_statement
    ;

do_statement:
    RW_DO INTEGER IDENTIFIER '=' expression_list
    ;

logical_if_statement:
    if_expression subset_executable_statement
    ;

if_expression:
    RW_IF '(' expression ')'
    ;

block_if_statement:
    RW_IF '(' expression ')' RW_THEN
    ;

else_statement:
    RW_ELSE
    ;

else_if_statement:
    RW_ELSE_IF '(' expression ')' RW_THEN
    ;

end_if_statement:
    RW_END_IF
    ;

subset_executable_statement:
    assignment_statement
    |
    assign_statement
    |
    arithmetic_if_statement
    |
    continue_statement
    |
    call_statement
    |
    return_statement
    |
    unconditional_go_to_statement
    |
    computed_go_to_statement
    |
    assigned_go_to_statement
    |
    stop_statement
    |
    pause_statement
    |
    io_statement
    ;

```



```

assignment_statement:
    variable '=' expression
    ;

assign_statement:
    RW_ASSIGN INTEGER RW_TO IDENTIFIER
    ;

arithmetic_if_statement:
    RW_IF '(' expression ')' integer_list
    ;

continue_statement:
    RW_CONTINUE
    ;

call_statement:
    RW_CALL IDENTIFIER
    {
        add_list( &subprogram_list, $2, 0, 0 );
    }
    |
    RW_CALL IDENTIFIER optional_actual_argument_list
    {
        add_list( &subprogram_list, $2, 0, 0 );
    }
    ;

optional_actual_argument_list:
    '(' ',' ')'
    |
    '(' actual_argument_list ')'
    ;

actual_argument_list:
    actual_argument
    |
    actual_argument_list ',' actual_argument
    ;

actual_argument:
    expression
    |
    actual_argument_alterate_return
    ;

actual_argument_alterate_return:
    '*' INTEGER
    ;

return_statement:
    RW_RETURN optional_expression
    ;

unconditional_go_to_statement:
    RW_GO_TO INTEGER
    ;

computed_go_to_statement:
    RW_GO_TO '(' integer_list ')' optional_comma expression
    ;

assigned_go_to_statement:
    RW_GO_TO IDENTIFIER
    |
    RW_GO_TO IDENTIFIER optional_comma '(' integer_list ')'
    ;

```

```

optional_comma:
    /* NULL */
    |
    | ','
    ;

integer_list:
    INTEGER
    |
    | integer_list ',' INTEGER
    ;

pause_statement:
    RW_PAUSE optional_expression
    ;

stop_statement:
    RW_STOP optional_expression
    ;

io_statement:
    open_statement
    |
    | close_statement
    |
    | inquire_statement
    |
    | read_statement
    |
    | write_statement
    |
    | print_statement
    |
    | backspace_statement
    |
    | rewind_statement
    |
    | endfile_statement
    ;

open_statement:
    RW_OPEN '(' control_information_list ')'
    ;

close_statement:
    RW_CLOSE '(' control_information_list ')'
    ;

inquire_statement:
    RW_INQUIRE '(' control_information_list ')'
    ;

read_statement:
    RW_READ '(' control_information_list ')' optional_io_list
    |
    | RW_READ control
    |
    | RW_READ control ',' io_list
    ;

write_statement:
    RW_WRITE '(' control_information_list ')' optional_io_list
    ;

print_statement:
    RW_PRINT control
    |
    | RW_PRINT control ',' io_list
    ;

```

```

backspace_statement:
    RW_BACKSPACE '(' control_information_list ')'
    |
    RW_BACKSPACE control
    ;

rewind_statement:
    RW_REWIND '(' control_information_list ')'
    |
    RW_REWIND control
    ;

endfile_statement:
    RW_ENDFILE '(' control_information_list ')'
    |
    RW_ENDFILE control
    ;

control_information_list:
    control_information
    |
    control_information_list ',' control_information
    ;

control_information:
    control
    |
    IDENTIFIER '=' expression
    ;

control:
    variable
    |
    constant
    |
    '*'
    ;

optional_io_list:
    /* NULL */
    |
    io_list
    ;

io_list:
    io
    |
    io_list ',' io
    ;

io:
    expression
    |
    io_implied_do_list
    ;

io_implied_do_list:
    '(' io_list ',' IDENTIFIER '=' expression_list ')'
    ;

format_statement:
    RW_FORMAT
    ;

%%

FILE: structure/include/list.h

```

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#define LIST struct list_type
LIST
{
    char *identifier;
    int type;
    LIST *call_list;
    LIST *next;
};

extern LIST *end_list( );
extern LIST *add_list( );
extern LIST *find_list( );
extern void print_list( );
extern void delete_list( );

FILE: structure/library/Makefile

#
# Copyright 1991
# Georgia Institute of Technology
# Computer Engineering Research Laboratory
# Author: Stephen R. Wachtel
#

CC = cc -g
INCLUDE = ../include
CFLAGS = -I$(INCLUDE)
LIBRARY = library.a

OBJECTS = \
    duplicate.o \
    hollerith.o \
    link_list.o \
    lowercase.o \
    main.o \
    non_blank.o \
    summary.o \
    yyerror.o \
    yygetc.o \
    yywrap.o

$(LIBRARY): $(OBJECTS)
    ar crv $(LIBRARY) $(OBJECTS)
    ranlib $(LIBRARY)

.SUFFIXES: .c .o
.c.o:
    $(CC) -c $(CFLAGS) $<

clean:
    rm -f $(LIBRARY) $(OBJECTS)

FILE: structure/library/duplicate.c

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include <stdio.h>

```

```

#include <string.h>
#include <malloc.h>

char *duplicate( string )
register char *string;
{
    register char *temporary = (char *)NULL;

    if ( string != (char *)NULL )
    {
        if ( ( temporary = (char *)malloc( strlen( string ) + 1 ) ) != (char *)NULL )
            strcpy( temporary, string );
        else
            fprintf( stderr, "ERROR: duplicate( %s )\n", string );
    }

    return( temporary );
} /* duplicate */

```

FILE: structure/library/hollerith.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include <stdio.h>

char *hollerith( string, delimiter )
register char *string;
register char delimiter;
{
    int hollerith_length;
    register int string_length = 0;

    sscanf( string, "%dh", &hollerith_length );

    string[ string_length++ ] = delimiter;
    while ( hollerith_length != 0 )
    {
        if ( ( string[ string_length ] = yyinput( ) ) == '\n' )
        {
            yyunput( string[ string_length ] );
            break;
        }

        string_length++;
        hollerith_length--;
    }
    string[ string_length++ ] = delimiter;

    string[ string_length ] = '\0';
    return( string );
} /* hollerith */

```

FILE: structure/library/link\_list.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include <stdio.h>
#include <malloc.h>
#include "list.h"

LIST *end_list( list )
register LIST *list;

```

```

{
    if ( list != (LIST *)NULL )
    {
        while ( list->next != (LIST *)NULL )
            list = list->next;
    }

    return( list );
} /* end_list */

LIST *add_list( list, identifier, type, call_list )
register LIST **list;
register char *identifier;
register int type;
register LIST *call_list;
{
    register LIST *temporary = (LIST *)malloc( sizeof( LIST ) );

    temporary->identifier = identifier;
    temporary->type = type;
    temporary->call_list = call_list;
    temporary->next = (LIST *)NULL;

    if ( *list == (LIST *)NULL )
        *list = temporary;
    else
        end_list( *list )->next = temporary;

    return( temporary );
} /* add_list */

LIST *find_list( list, identifier )
register LIST *list;
register char *identifier;
{
    while ( list != (LIST *)NULL )
    {
        if ( strcmp( list->identifier, identifier ) == 0 )
            return( list );

        list = list->next;
    }

    return( (LIST *)NULL );
} /* find_list */

void print_list( file, list )
register FILE *file;
register LIST *list;
{
    register LIST *call_list;

    while ( list != (LIST *)NULL )
    {
        fprintf( file, "%s:%d\n", list->identifier, list->type );

        call_list = list->call_list;
        while ( call_list != (LIST *)NULL )
        {
            fprintf( file, "\t%s\n", call_list->identifier );
            call_list = call_list->next;
        }

        list = list->next;
    }
} /* print_list */

void delete_list( list )
register LIST *list;
{
    if ( list != (LIST *)NULL )
    {
        delete_list( list->next );

        free( list );
    }
}

```

```

} /* delete_list */

```

```

FILE: structure/library/lowercase.c

```

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

char *lowercase( string )
register char *string;
{
    register int index = 0;

    while ( string[ index ] != '\0' )
    {
        string[ index ] = tolower( string[ index ] );
        index++;
    }

    return( string );
} /* lowercase */

```

```

FILE: structure/library/main.c

```

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```

#include <stdio.h>

```

```

extern FILE *yyin;
extern FILE *yyout;

```

```

#define PROGRAM argument[ 0 ]
#define INPUT_FILE argument[ 1 ]
#define OUTPUT_FILE argument[ 2 ]

```

```

int main( number_argument, argument )
int number_argument;
char *argument[ ];

```

```

{
    if ( number_argument == 1 )
    {
        yyin = stdin;
        yyout = stdout;

        yyparse( );
        exit( 0 );
    }

    if ( number_argument == 3 )
    {
        if ( ( yyin = fopen( INPUT_FILE, "r" ) ) == (FILE *)NULL )
        {
            fprintf( stderr, "%s: ERROR - unable to open input file '%s' n", PROGRAM,
INPUT_FILE );
            exit( -1 );
        }

        if ( ( yyout = fopen( OUTPUT_FILE, "w" ) ) == (FILE *)NULL )
        {
            fprintf( stderr, "%s: ERROR - unable to open output file '%s'\n", PROGRAM,
OUTPUT_FILE );
            exit( -1 );
        }
    }
}

```

```

        yyparse( );
        exit( 0 );
    }

    fprintf( stderr, "usage: %s <input file> <output file>\n", PROGRAM );
    exit( 0 );
} /* main */

```

FILE: structure/library/non\_blank.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include <string.h>

char *non_blank( string )
register char *string;
{
    register int offset;
    register int length;

    length = strlen( string ) - 1;
    while ( ( string[ length ] == ' ' ) && ( string[ length ] != '\0' ) )
        string[ length-- ] = '\0';

    offset = 0;
    while ( ( string[ offset ] == ' ' ) && ( string[ offset ] != '\0' ) )
        string[ offset++ ] = '\0';

    strcpy( string, &string[ offset ] );

    if ( strlen( string ) != 0 )
        return( string );
    else
        return( 0 );
} /* non_blank */

```

FILE: structure/library/summary.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include <stdio.h>
#include <malloc.h>
#include "grammar.h"
#include "list.h"

extern FILE *yyin;
extern FILE *yyout;

void print_level( level )
register int level;
{
    while ( level-- != 0 )
        fprintf( yyout, "\t" );
} /* print_level */

void print_trace( block_list, identifier, level )
LIST *block_list;
char *identifier;
int level;
{
    LIST *list;

```



```

LIST *call_list;
#define TRACE 0x10000000

print_level( level );
fprintf( yyout, "%s\n", identifier );

if ( ( list = find_list( block_list, identifier ) ) == (LIST *)NULL )
    return;

if ( ( list->type & TRACE ) != TRACE )
{
    list->type |= TRACE;

    call_list = list->call_list;
    while ( call_list != (LIST *)NULL )
    {
        print_trace( block_list, call_list->identifier, level + 1 );

        call_list = call_list->next;
    }

    list->type &= ~TRACE;
}
/* print_trace */

void summary( list )
register LIST *list;
{
    while ( list != (LIST *)NULL )
    {
        if ( list->type == RW_PROGRAM )
            print_trace( list, list->identifier, 0 );

        list = list->next;
    }

    exit( 0 );
} /* summary */

```

FILE: structure/library/yyerror.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include <stdio.h>

extern int yylineno;

void yyerror( string )
register char *string;
{
    fprintf( stderr, "line %d, %s\n", yylineno, string );

    exit( -1 );
} /* yyerror */

```

FILE: structure/library/yygetc.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include <stdio.h>
#include <ctype.h>

```

```

extern int yylineno;

int tab( length )
register int length;
{
    while ( length-- != 0 )
        yyunput( ' ' );

    return( ' ' );
} /* tab */

int yygetc( file )
register FILE *file;
{
    int c;
    int column[ 6 ];

loop:
    if ( ( c = getc( file ) ) == '\t' )
        c = tab( 6 );

    if ( c != '\n' )
        return( c );

    if ( ( column[ 0 ] = getc( file ) ) != ' ' )
        goto abort_0;
    if ( ( column[ 1 ] = getc( file ) ) != ' ' )
        goto abort_1;
    if ( ( column[ 2 ] = getc( file ) ) != ' ' )
        goto abort_2;
    if ( ( column[ 3 ] = getc( file ) ) != ' ' )
        goto abort_3;
    if ( ( column[ 4 ] = getc( file ) ) != ' ' )
        goto abort_4;
    if ( isspace( column[ 5 ] = getc( file ) ) )
        goto abort_5;

    yylineno++;
    goto loop;

abort_5:
    if ( column[ 5 ] == '\t' )
        tab( 1 );
    else
    {
        yyunput( column[ 5 ] );
        if ( column[ 5 ] == '\n' )
            yylineno++;
    }

abort_4:
    if ( column[ 4 ] == '\t' )
        tab( 2 );
    else
    {
        yyunput( column[ 4 ] );
        if ( column[ 4 ] == '\n' )
            yylineno++;
    }

abort_3:
    if ( column[ 3 ] == '\t' )
        tab( 3 );
    else
    {
        yyunput( column[ 3 ] );
        if ( column[ 3 ] == '\n' )
            yylineno++;
    }

abort_2:
    if ( column[ 2 ] == '\t' )
        tab( 4 );
    else
    {
        yyunput( column[ 2 ] );
        if ( column[ 2 ] == '\n' )
            yylineno++;
    }

```

```

    }
abort_1:
    if ( column[ 1 ] == '\t' )
        tab( 5 );
    else
    {
        yyunput( column[ 1 ] );
        if ( column[ 1 ] == '\n' )
            yylineno++;
    }
abort_0:
    if ( column[ 0 ] == '\t' )
        tab( 6 );
    else
    {
        yyunput( column[ 0 ] );
        if ( column[ 0 ] == '\n' )
            yylineno++;
    }
    return( c );
} /* yygetc */

```

FILE: structure/library/yywrap.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

int yywrap( )
{
    return( 1 );
} /* yywrap */

```

FILE: structure/scanner.l

```

%{
/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */
%}

%a 10000
%b 10000
%c 10000
%d 10000
%e 10000
%f 10000
%g 10000
%h 10000
%i 10000
%j 10000
%k 10000
%l 10000
%m 10000
%n 10000
%o 10000
%p 10000

a  [aA]
b  [bB]
c  [cC]
d  [dD]
e  [eE]
f  [fF]
g  [gG]
h  [hH]
i  [iI]
j  [jJ]
k  [kK]
l  [lL]
m  [mM]
n  [nN]
o  [oO]
p  [pP]
q  [qQ]

```

```

r  [rR]
s  [sS]
t  [tT]
u  [uU]
v  [vV]
w  [wW]
x  [xX]
y  [yY]
z  [zZ]

%{
#include "grammar.h"
extern char *yylval;

#undef YYLMAX
#define YYLMAX (256*20)

extern char *duplicate();
extern char *hollerith();
extern char *non_blanks();
extern char *lowercase();
%}

%%

^([\\cC].*[\\n]) |
^([\\ ])*[\\n] {
#ifdef DEBUG
    ECHO;
#endif
    yyval = duplicate( yytext );
    return( COMMENT );
}

[\\ ] {
#ifdef DEBUG
    ECHO;
#endif
    /* return( '\\ ' ) */;
}

[\\&] {
#ifdef DEBUG
    ECHO;
#endif
    return( '\\&' );
}

[\\(] {
#ifdef DEBUG
    ECHO;
#endif
    return( '\\(' );
}

[\\)] {
#ifdef DEBUG
    ECHO;
#endif
    return( '\\)' );
}

[\\*] {
#ifdef DEBUG
    ECHO;
#endif
    return( '\\*' );
}

[\\*][\\*] {

```

```

#ifdef DEBUG
    ECHO;
#endif
    return( EXPONENTIATE );
}

```

```

[\\+] {
#ifdef DEBUG
    ECHO;
#endif
    return( '\\+' );
}

```

```

[\\,] {
#ifdef DEBUG
    ECHO;
#endif
    return( '\\,' );
}

```

```

[\\-] {
#ifdef DEBUG
    ECHO;
#endif
    return( '\\-' );
}

```

```

[\\.] {
#ifdef DEBUG
    ECHO;
#endif
    return( '\\.' );
}

```

```

[\\/] {
#ifdef DEBUG
    ECHO;
#endif
    return( '\\/' );
}

```

```

[\\:] {
#ifdef DEBUG
    ECHO;
#endif
    return( '\\:' );
}

```

```

[\\=] {
#ifdef DEBUG
    ECHO;
#endif
    return( '\\=' );
}

```

```

[\\n] {
#ifdef DEBUG
    ECHO;
#endif
    /* return( '\\n' ) */;
}

```

```

[\\t] {
#ifdef DEBUG
    ECHO;
#endif
    /* return( '\\t' ) */;
}

```

```

[\\.]{a}{n}{d}[\\.] {
#ifdef DEBUG

```

```

    ECHO;
#endif
    return( RW_AND );
}

[\\.]{e}{q}[\\.] {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_EQ );
}

[\\.]{e}{q}{v}[\\.] {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_EQV );
}

[\\.]{f}{a}{l}{s}{e}[\\.] {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_FALSE );
}

[\\.]{g}{e}[\\.] {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_GE );
}

[\\.]{g}{t}[\\.] {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_GT );
}

[\\.]{l}{e}[\\.] {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_LE );
}

[\\.]{l}{t}[\\.] {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_LT );
}

[\\.]{n}{e}[\\.] {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_NE );
}

[\\.]{n}{e}{q}{v}[\\.] {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_NEQV );
}

[\\.]{n}{o}{t}[\\.] {
#ifdef DEBUG
    ECHO;

```

```

#endif
    return( RW_NOT );
}

[\\.]{}{o}{r}{\\.} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_OR );
}

[\\.]{}{t}{r}{u}{e}{\\.} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_TRUE );
}

{a}{s}{s}{i}{g}{n} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_ASSIGN );
}

{b}{a}{c}{k}{s}{p}{a}{c}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_BACKSPACE );
}

{b}{l}{o}{c}{k}{[\\ ]*(d){a}{t}{a}} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_BLOCK_DATA );
}

{c}{a}{l}{l} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_CALL );
}

{c}{h}{a}{r}{a}{c}{t}{e}{r} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_CHARACTER );
}

{c}{l}{o}{s}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_CLOSE );
}

{c}{o}{m}{m}{o}{n} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_COMMON );
}

{c}{o}{m}{p}{l}{e}{x} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_COMPLEX );
}

```

```

    return( RW_COMPLEX );
}

{c}{o}{n}{t}{i}{n}{u}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_CONTINUE );
}

{d}{a}{t}{a} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_DATA );
}

{d}{i}{m}{e}{n}{s}{i}{o}{n} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_DIMENSION );
}

{d}{o} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_DO );
}

{d}{o}{u}{b}{l}{e}{[ \ ]*{p}{r}{e}{c}{i}{s}{i}{o}{n} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_DOUBLE_PRECISION );
}

{e}{l}{s}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_ELSE );
}

{e}{l}{s}{e}{[ \ ]*{i}{f} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_ELSE_IF );
}

{e}{n}{d} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_END );
}

{e}{n}{d}{[ \ ]*{i}{f} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_END_IF );
}

{e}{n}{d}{f}{i}{l}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_ENDFILE );
}

```



```

    }

    {e}{n}{t}{r}{y} {
#ifdef DEBUG
        ECHO;
#endif
        return( RW_ENTRY );
    }

    {e}{q}{u}{i}{l}{v}{a}{l}{e}{n}{c}{e} {
#ifdef DEBUG
        ECHO;
#endif
        return( RW_EQUIVALENCE );
    }

    {e}{x}{t}{e}{r}{n}{a}{l} {
#ifdef DEBUG
        ECHO;
#endif
        return( RW_EXTERNAL );
    }

    {f}{o}{r}{m}{a}{t}.* {
#ifdef DEBUG
        ECHO;
#endif
        yylval = duplicate( yytext );
        return( RW_FORMAT );
    }

    {f}{u}{n}{c}{t}{i}{o}{n} {
#ifdef DEBUG
        ECHO;
#endif
        return( RW_FUNCTION );
    }

    {g}{o}{\ }*(t){o} {
#ifdef DEBUG
        ECHO;
#endif
        return( RW_GO_TO );
    }

    {i}{f} {
#ifdef DEBUG
        ECHO;
#endif
        return( RW_IF );
    }

    {i}{m}{p}{l}{i}{c}{i}{t} {
#ifdef DEBUG
        ECHO;
#endif
        return( RW_IMPLICIT );
    }

    {i}{n}{c}{l}{u}{d}{e} {
#ifdef DEBUG
        ECHO;
#endif
        return( RW_INCLUDE );
    }

    {i}{n}{q}{u}{i}{r}{e} {
#ifdef DEBUG
        ECHO;
#endif
        return( RW_INQUIRE );
    }

```

```

    }

    {i}{n}{t}{e}{g}{e}{r} {
#ifdef DEBUG
        ECHO;
#endif
        return( RW_INTEGER );
    }

    {i}{n}{t}{r}{i}{n}{s}{i}{c} {
#ifdef DEBUG
        ECHO;
#endif
        return( RW_INTRINSIC );
    }

    {l}{o}{g}{i}{c}{a}{l} {
#ifdef DEBUG
        ECHO;
#endif
        return( RW_LOGICAL );
    }

    {n}{a}{m}{e}{l}{i}{s}{t} {
#ifdef DEBUG
        ECHO;
#endif
        return( RW_NAMELIST );
    }

    {o}{p}{e}{n} {
#ifdef DEBUG
        ECHO;
#endif
        return( RW_OPEN );
    }

    {p}{a}{r}{a}{m}{e}{t}{e}{r} {
#ifdef DEBUG
        ECHO;
#endif
        return( RW_PARAMETER );
    }

    {p}{a}{u}{s}{e} {
#ifdef DEBUG
        ECHO;
#endif
        return( RW_PAUSE );
    }

    {p}{r}{i}{n}{t} {
#ifdef DEBUG
        ECHO;
#endif
        return( RW_PRINT );
    }

    {p}{r}{o}{g}{r}{a}{m} {
#ifdef DEBUG
        ECHO;
#endif
        return( RW_PROGRAM );
    }

    {r}{e}{a}{d} {
#ifdef DEBUG
        ECHO;
#endif
        return( RW_READ );
    }

```

```

{r}{e}{a}{l} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_REAL );
}

{r}{e}{t}{u}{r}{n} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_RETURN );
}

{r}{e}{w}{i}{n}{d} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_REWIND );
}

{s}{a}{v}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_SAVE );
}

{s}{t}{o}{p} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_STOP );
}

{s}{u}{b}{r}{o}{u}{t}{i}{n}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_SUBROUTINE );
}

{t}{h}{e}{n} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_THEN );
}

{t}{o} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_TO );
}

{w}{r}{i}{t}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_WRITE );
}

{u}{n}{d}{e}{f}{i}{n}{e}{d} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_UNDEFINED );
}

```

```

[%a-zA-Z][a-zA-Z0-9]* {
#ifdef DEBUG
    ECHO;
#endif
    yylval = duplicate( lowercase( yytext ) );
    return( IDENTIFIER );
}

^[0-9][0-9][0-9][0-9][0-9][\ ] {
#ifdef DEBUG
    ECHO;
#endif
    yylval = duplicate( non_blank( yytext ) );
    return( LABEL );
}

[0-9]+ |
[0-9]+/\.[a-zA-Z]+\.\. {
#ifdef DEBUG
    ECHO;
#endif
    yylval = duplicate( yytext );
    return( INTEGER );
}

[0-9]+\.[0-9]*([eE][\+\-]?[0-9]+)? |
[0-9]*\.[0-9]+([eE][\+\-]?[0-9]+)? |
[0-9]+([eE][\+\-]?[0-9]+)? {
#ifdef DEBUG
    ECHO;
#endif
    yylval = duplicate( yytext );
    return( REAL );
}

[0-9]+\.[0-9]*([dD][\+\-]?[0-9]+)? |
[0-9]*\.[0-9]+([dD][\+\-]?[0-9]+)? |
[0-9]+([dD][\+\-]?[0-9]+)? {
#ifdef DEBUG
    ECHO;
#endif
    yylval = duplicate( yytext );
    return( DOUBLE_PRECISION );
}

\[^\']**\' |
\[^\"]*\" {
#ifdef DEBUG
    ECHO;
#endif
    yytext[ 0 ] = '\\';
    yytext[ strlen( yytext ) - 1 ] = '\\';
    yylval = duplicate( yytext );
    return( STRING );
}

[0-9]+[hH] {
#ifdef DEBUG
    ECHO;
#endif
    yylval = duplicate( hollerith( yytext, '\\\"' ) );
    return( HOLLERITH );
}

```

**18. Appendix M: usage program source**

FILE: usage/combine/Makefile

```
#
# Copyright 1991
# Georgia Institute of Technology
# Computer Engineering Research Laboratory
# Author: Stephen R. Wachtel
#
```

```
default:    combine
```

```
CC = cc -g
INCLUDE = include
CFLAGS = -I$(INCLUDE)
LIBRARY = library/library.a
```

```
combine.o: combine.c
    $(CC) $(CFLAGS) -c combine.c
```

```
combine:    combine.o $(LIBRARY)
    $(CC) -o combine combine.o $(LIBRARY)
```

```
clean:
    cd library; make clean
    rm -f combine combine.o
```

FILE: usage/combine/combine.c

```
/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */
```

```
#include <stdio.h>
#include "table.h"
#include "list.h"
```

```
extern char *parse( );
```

```
void input_list( processor_number, file_name )
register int processor_number;
register char *file_name;
{
    register FILE *file;
    char identifier[ 256 ];
    char usage;

    if ( ( file = fopen( file_name, "r" ) ) == (FILE *)NULL )
    {
        fprintf( stderr, "ERROR: unable to open for read '%s'\n", file_name );
        exit( -1 );
    }

    while ( fscanf( file, "%s %c\n", identifier, &usage ) == 2 )
        add_list( identifier )->usage[ processor_number ] = usage;

    fclose( file );
} /* input_list */
```

```
int network_transfer( usage )
register char *usage;
{
    register int s = 0;
```

```

register int r = 0;
while ( *usage != '\0' )
{
    switch ( *usage )
    {
        case 'S':
            s++;
            break;

        case 'R':
            r++;
            break;
    }

    usage++;
}

return( ( s > 1 ) || ( ( s == 1 ) && ( r != 0 ) ) );
} /* network_transfer */

int length_list( list )
register LIST *list;
{
    register int length = 0;
    register TABLE *table;

    while ( list != (LIST *)NULL )
    {
        if ( network_transfer( list->usage ) != 0 )
        {
            if ( ( table = find_table( list->identifier ) ) == (TABLE *)NULL )
            {
                fprintf( stderr, "ERROR: '%s' not found in table\n", list->identifier );
                exit( -1 );
            }

            length += length_subscript_list( table->subscript_list );
        }

        list = list->next;
    }

    return( length );
} /* length_list */

void output_variable( file, identifier, type, subscript_list, usage )
register FILE *file;
register char *identifier;
register char *type;
register char *subscript_list;
register char *usage;
{
    char *subscript;
    int lower[ 2 ];
    int upper[ 2 ];
    int index[ 2 ];

    if ( ( subscript = parse( subscript_list ) ) == (char *)NULL )
    {
        lower[ 0 ] = 0;
        upper[ 0 ] = 0;
    }
    else
    {
        lower[ 0 ] = atoi( parse( subscript ) );
        upper[ 0 ] = atoi( parse( subscript ) );
    }

    if ( ( subscript = parse( subscript_list ) ) == (char *)NULL )
    {
        lower[ 1 ] = 0;
        upper[ 1 ] = 0;
    }
    else
    {
        lower[ 1 ] = atoi( parse( subscript ) );
        upper[ 1 ] = atoi( parse( subscript ) );
    }
}

```

```

if ( ( subscript = parse( subscript_list ) ) != (char *)NULL )
{
    fprintf( stderr, "ERROR: output_variable( %s )\n", identifier );
    exit( -1 );
}

if ( ( lower[ 0 ] != 0 ) && ( upper[ 0 ] != 0 ) )
{
    index[ 0 ] = lower[ 0 ];
    while ( index[ 0 ] <= upper[ 0 ] )
    {
        if ( ( lower[ 1 ] != 0 ) && ( upper[ 1 ] != 0 ) )
        {
            index[ 1 ] = lower[ 1 ];
            while ( index[ 1 ] <= upper[ 1 ] )
            {
                fprintf( file, "%s(%0*d,%0*d) %s %s\n", identifier, digit( upper[ 0 ]
), index[ 0 ], digit( upper[ 1 ] ), index[ 1 ], type, usage );
                index[ 1 ]++;
            }
        }
        else
        {
            fprintf( file, "%s(%0*d) %s %s\n", identifier, digit( upper[ 0 ] ), index[
0 ], type, usage );
        }
        index[ 0 ]++;
    }
}
else
    fprintf( file, "%s %s %s\n", identifier, type, usage );
} /* output_variable */

void output_list( file )
register FILE *file;
{
    register TABLE *table;

    fprintf( file, "%d\n", length_list( list ) );

    while ( list != (LIST *)NULL )
    {
        if ( network_transfer( list->usage ) != 0 )
        {
            if ( ( table = find_table( list->identifier ) ) == (TABLE *)NULL )
            {
                fprintf( stderr, "ERROR: '%s' not found in table\n", list->identifier );
                exit( -1 );
            }

            output_variable( file, list->identifier, table->type, table->subscript_list,
list->usage );
        }

        list = list->next;
    }
} /* output_list */

#define PROGRAM argument[ 0 ]
#define ARGUMENT argument[ argument_number ]

int main( number_argument, argument )
register int number_argument;
register char *argument[ ];
{
    register int argument_number = 0;
    int processor_number;
    char file_name[ 256 ];

    if ( --number_argument == 0 )
    {
        fprintf( stderr, "usage: %s 00=<file name>...31=<file name>\n", PROGRAM );
        exit( 0 );
    }

    while ( argument_number++ != number_argument )
    {

```

```

    if ( sscanf( ARGUMENT, "%d=%s", &processor_number, file_name ) != 2 )
    {
        fprintf( stderr, "ERROR: unable to parse argument '%s'\n", ARGUMENT );
        exit( -1 );
    }

    input_list( processor_number, file_name );
}

initialize_table( stdin );
output_list( stdout );

exit( 0 );
} /* main */

```

FILE: usage/combine/include/list.h

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```

#define LIST struct list_type
LIST
{
    char *identifier;
    char *usage;
    LIST *next;
};

```

```

extern LIST *end_list( );
extern LIST *add_end_list( );
extern LIST *add_list( );
extern LIST *find_list( );

```

```

extern LIST *list;

```

FILE: usage/combine/include/table.h

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```

#define TABLE struct table_type
TABLE
{
    char *identifier;
    char *type;
    char *subscript_list;
    int class;
};

```

```

extern void allocate_table( );
extern void input_table( );
extern int compare( );
extern void sort_table( );
extern void initialize_table( );
extern TABLE *find_table( );

```

```

extern TABLE *table;
extern int number_table;

```

FILE: usage/combine/library/Makefile



```
#
# Copyright 1991
# Georgia Institute of Technology
# Computer Engineering Research Laboratory
# Author: Stephen R. Wachtel
#
```

```
CC = cc -g
INCLUDE = ../include
CFLAGS = -I$(INCLUDE)
LIBRARY = library.a
```

```
OBJECTS = \
    digit.o \
    duplicate.o \
    length_subscript_list.o \
    link_list.o \
    parse.o \
    table.o
```

```
$(LIBRARY): $(OBJECTS)
    rm -f $(LIBRARY)
    ar crv $(LIBRARY) $(OBJECTS)
    ranlib $(LIBRARY)
```

```
.SUFFIXES: .c .o
.c.o:
    $(CC) -c $(CFLAGS) $<
```

```
clean:
    rm -f $(LIBRARY) $(OBJECTS)
```

```
FILE: usage/combine/library/digit.c
```

```
/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */
```

```
int digit( number )
register int number;
{
    register int count = 0;

    do
    {
        number /= 10;
        count++;
    } while ( number != 0 );

    return( count );
} /* digit */
```

```
FILE: usage/combine/library/duplicate.c
```

```
/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */
```

```
#include <stdio.h>
#include <string.h>
#include <malloc.h>
```

```
char *duplicate( string )
```

```

register char *string;
{
    register char *temporary = (char *)NULL;

    if ( string != (char *)NULL )
    {
        if ( ( temporary = (char *)malloc( strlen( string ) + 1 ) ) != (char *)NULL )
            strcpy( temporary, string );
        else
            fprintf( stderr, "ERROR: duplicate( %s )\n", string );
    }

    return( temporary );
} /* duplicate */

```

FILE: usage/combine/library/length\_subscript\_list.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

extern char *duplicate( );
extern char *parse( );

int length_subscript_list( subscript_list )
register char *subscript_list;
{
    register char *subscript;
    register int lower;
    register int upper;
    register int length = 1;

    subscript_list = duplicate( subscript_list );

    while ( ( subscript = parse( subscript_list ) ) != 0 )
    {
        lower = atoi( parse( subscript ) );
        upper = atoi( parse( subscript ) );
        length *= ( upper - lower + 1 );
    }

    return( length );
} /* length_subscript_list */

```

FILE: usage/combine/library/link\_list.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include <stdio.h>
#include <malloc.h>
#include <string.h>
#include "list.h"

extern char *duplicate( );

LIST *list = (LIST *)NULL;

LIST *end_list( list )
register LIST *list;
{
    if ( list != (LIST *)NULL )
    {
        while ( list->next != (LIST *)NULL )

```

```

        list = list->next;
    }

    return( list );
} /* end_list */

LIST *add_end_list( list, identifier, usage )
register LIST **list;
register char *identifier;
register char *usage;
{
    register LIST *temporary = (LIST *)malloc( sizeof( LIST ) );

    temporary->identifier = duplicate( identifier );
    temporary->usage = duplicate( usage );
    temporary->next = (LIST *)NULL;

    if ( *list == (LIST *)NULL )
        *list = temporary;
    else
        end_list( *list )->next = temporary;

    return( temporary );
} /* add_end_list */

LIST *add_list( identifier )
register Char *identifier;
{
    register LIST *temporary = find_list( list, identifier );

    if ( temporary == (LIST *)NULL )
        temporary = add_end_list( &list, identifier, "-----" );

    return( temporary );
} /* add_list */

LIST *find_list( list, identifier )
register LIST *list;
register char *identifier;
{
    while ( list != (LIST *)NULL )
    {
        if ( strcmp( list->identifier, identifier ) == 0 )
            return( list );

        list = list->next;
    }

    return( (LIST *)NULL );
} /* find_list */

```

FILE: usage/combine/library/parse.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachuel
 */

```

```

#include <string.h>

```

```

extern char *duplicate( );

```

```

char *parse( list )
register char *list;
{
    register int length = 0;
    register int brace = 0;
    register char *temporary = (char *)0;

    for (;;)
    {

```

```

        switch ( list[ length ] )
        {
            case '{':
                brace++;
                break;

            case '}':
                brace--;
                break;
        }

        if ( brace == 0 )
            break;

        length++;
    }

    if ( length != 0 )
    {
        list[ length ] = '\0';
        temporary = duplicate( list + 1 );
        strcpy( list, list + 1 + length );
    }
    else
    {
        if ( list[ length ] != '\0' )
        {
            temporary = duplicate( list );
            list[ length ] = '\0';
        }
    }

    return( temporary );
} /* parse */

```

FILE: usage/combine/library/table.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include <stdio.h>
#include <malloc.h>
#include <string.h>
#include "table.h"

extern char *duplicate( );

TABLE *table = (TABLE *)NULL;
int number_table = 0;

void allocate_table( )
{
    if ( ( table = (TABLE *)malloc( sizeof( TABLE ) * number_table ) ) == (TABLE *)NULL )
    {
        fprintf( stderr, "ERROR: unable to allocate 'table'\n" );
        exit( -1 );
    }
} /* allocate_table */

void input_table( file )
register FILE *file;
{
    int table_number;
    char identifier[ 256 ];
    char type[ 256 ];
    char subscript_list[ 256 ];
    int class;

    for ( table_number = 0; table_number != number_table; table_number++ )
    {

```

```

4 )      if ( fscanf( file, "%s %s %s %x\n", identifier, type, subscript_list, &class ) !=
         {
              fprintf( stderr, "ERROR: unable to read 'table[ %s ]'\n", table_number );
              exit( -1 );
         }

         table[ table_number ].identifier = duplicate( identifier );
         table[ table_number ].type = duplicate( type );
         table[ table_number ].subscript_list = duplicate( subscript_list );
         table[ table_number ].class = class;
         }
} /* input_table */

int compare( table1, table2 )
TABLE *table1;
TABLE *table2;
{
    return( strcmp( table1->identifier, table2->identifier ) );
} /* compare */

void sort_table( )
{
    qsort( table, number_table, sizeof( TABLE ), compare );
} /* sort_table */

void initialize_table( file )
register FILE *file;
{
    fscanf( file, "%d\n", &number_table );
    allocate_table( );
    input_table( file );
    sort_table( );
} /* initialize_table */

TABLE *find_table( identifier )
register char *identifier;
{
    register int low, high;
    register int middle, test;

    low = 0;
    high = number_table - 1;

    while ( low <= high )
    {
        middle = ( low + high ) / 2;
        test = strcmp( identifier, table[ middle ].identifier );

        if ( test < 0 )
        {
            high = middle - 1;
            continue;
        }

        if ( test > 0 )
        {
            low = middle + 1;
            continue;
        }

        return( &table[ middle ] );
    }

    return( (TABLE *)NULL );
} /* find_table */

```

FILE: usage/sequence/Makefile

```

#
# Copyright 1991
# Georgia Institute of Technology
# Computer Engineering Research Laboratory
# Author: Stephen R. Wachtel
#

```

```

default:    sequence

CC = cc -g
INCLUDE = include
CFLAGS = -I$(INCLUDE)
LIBRARY = library/library.a

sequence.o: sequence.c
    $(CC) $(CFLAGS) -c sequence.c

sequence:  sequence.o $(LIBRARY)
    $(CC) -o sequence sequence.o $(LIBRARY)

clean:
    cd library; make clean
    rm -f sequence sequence.o

FILE: usage/sequence/include/table.h

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author:  Stephen R. Wachtel
 */

#define TABLE struct table_type
TABLE
{
    char *identifier;
    char *type;
    char *usage;
};

extern void allocate_table( );
extern void input_table( );
extern int compare( );
extern void sort_table( );
extern void initialize_table( );

extern TABLE *table;
extern int number_table;

FILE: usage/sequence/library/Makefile

#
# Copyright 1991
# Georgia Institute of Technology
# Computer Engineering Research Laboratory
# Author:  Stephen R. Wachtel
#

CC = cc -g
INCLUDE = ../include
CFLAGS = -I$(INCLUDE)
LIBRARY = library.a

OBJECTS = \
    count.o \
    duplicate.o \
    message_type.o \
    table.o \
    type_length.o

$(LIBRARY): $(OBJECTS)
    ar crv $(LIBRARY) $(OBJECTS)

```

```

ranlib $(LIBRARY)

.SUFFIXES: .c .o
.c.o:
    $(CC) -c $(CFLAGS) $<

clean:
    rm -f $(LIBRARY) $(OBJECTS)

FILE: usage/sequence/library/count.c

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```

int count( string, length, c )
register char *string;
register int length;
register char c;
{
    register int c_count = 0;

    while ( length != 0 )
    {
        if ( *string == c )
            c_count++;

        string++;
        length--;
    }

    return( c_count );
} /* count */

```

FILE: usage/sequence/library/duplicate.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```

#include <stdio.h>
#include <string.h>
#include <malloc.h>

```

```

char *duplicate( string )
register char *string;
{
    register char *temporary = (char *)NULL;

    if ( string != (char *)NULL )
    {
        if ( ( temporary = (char *)malloc( strlen( string ) + 1 ) ) != (char *)NULL )
            strcpy( temporary, string );
        else
            fprintf( stderr, "ERROR: duplicate( %s )\n", string );
    }

    return( temporary );
} /* duplicate */

```

FILE: usage/sequence/library/message\_type.c

```

/*
 * Copyright 1991

```

```

* Georgia Institute of Technology
* Computer Engineering Research Laboratory
* Author: Stephen R. Wachtel
*/

#include <stdio.h>

char *message_type( type )
register char *type;
{
    int length;

    if ( sscanf( type, "CHARACTER*d", &length ) == 1 )
        return( "CHARACTER_08BIT" );

    if ( sscanf( type, "COMPLEX*d", &length ) == 1 )
    {
        switch ( length )
        {
            case 8:
                return( "COMPLEX_32BIT" );

            case 16:
                return( "COMPLEX_64BIT" );

        }
    }

    if ( sscanf( type, "INTEGER*d", &length ) == 1 )
    {
        switch ( length )
        {
            case 1:
                return( "SIGNED_08BIT" );

            case 2:
                return( "SIGNED_16BIT" );

            case 4:
                return( "SIGNED_32BIT" );

        }
    }

    if ( sscanf( type, "LOGICAL*d", &length ) == 1 )
    {
        switch ( length )
        {
            case 1:
                return( "LOGICAL_08BIT" );

            case 2:
                return( "LOGICAL_16BIT" );

            case 4:
                return( "LOGICAL_32BIT" );

        }
    }

    if ( sscanf( type, "REAL*d", &length ) == 1 )
    {
        switch ( length )
        {
            case 4:
                return( "REAL_32BIT" );

            case 8:
                return( "REAL_64BIT" );

        }
    }

    fprintf( stderr, "ERROR: message_type( %s )\n", type );
    exit( -1 );
} /* message_type */

```

FILE: usage/sequence/library/table.c

```

/*
* Copyright 1991

```



```

* Georgia Institute of Technology
* Computer Engineering Research Laboratory
* Author: Stephen R. Wachtel
*/

```

```

#include <stdio.h>
#include <malloc.h>
#include <string.h>
#include "table.h"

```

```
extern char *duplicate( );
```

```
TABLE *table = (TABLE *)NULL;
int number_table = 0;
```

```

void allocate_table( )
{
    if ( ( table = (TABLE *)malloc( sizeof( TABLE ) * number_table ) ) == (TABLE *)NULL )
    {
        fprintf( stderr, "ERROR: unable to allocate 'table'\n" );
        exit( -1 );
    }
} /* allocate_table */

```

```

void input_table( file )
register FILE *file;
{
    register int table_number;
    char identifier[ 256 ];
    char type[ 256 ];
    char usage[ 256 ];

    for ( table_number = 0; table_number != number_table; table_number++ )
    {
        if ( fscanf( file, "%s %s %s\n", identifier, type, usage ) != 3 )
        {
            fprintf( stderr, "ERROR: unable to read 'table[ %d ]'\n", table_number );
            exit( -1 );
        }

        table[ table_number ].identifier = duplicate( identifier );
        table[ table_number ].type = duplicate( type );
        table[ table_number ].usage = duplicate( usage );
    }
} /* input_table */

```

```

int compare( table1, table2 )
register TABLE *table1;
register TABLE *table2;
{
    register int temporary = count( table1->usage, strlen( table1->usage ), '-' ) - count(
table2->usage, strlen( table2->usage ), '-' );

    if ( temporary != 0 )
        return( temporary );
    else
        return( strcmp( table1->identifier, table2->identifier ) );
} /* compare */

```

```

void sort_table( )
{
    qsort( table, number_table, sizeof( TABLE ), compare );
} /* sort_table */

```

```

void initialize_table( file )
register FILE *file;
{
    fscanf( file, "%d\n", &number_table );
    allocate_table( );
    input_table( file );
    sort_table( );
} /* initialize_table */

```

FILE: usage/sequence/library/type\_length.c

```
/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include <stdio.h>

int type_length( type )
register char *type;
{
    char name[256];
    static int length;

    if ( sscanf( type, "%[^']*%d", name, &length ) == 2 )
        return( length );
    else
        return( 0 );
} /* type_length */
```

FILE: usage/sequence/sequence.c

```
/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include <stdio.h>
#include <malloc.h>
#include <string.h>
#include "table.h"

extern char *duplicate( );
extern char *message_type( );

#define NUMBER_PROCESSOR 32
char *file_name[ NUMBER_PROCESSOR ];

static int cycle = 0;

int next_sender( last_sender, usage )
register int last_sender;
register char *usage;
{
    register int processor_number = last_sender;

    while ( usage[ ++processor_number ] != '\0' )
    {
        if ( usage[ processor_number ] == 'S' )
            return( processor_number );
    }

    return( -1 );
} /* next_sender */

int next_receiver( last_receiver, usage )
register int last_receiver;
register char *usage;
{
    register int processor_number = last_receiver;

    while ( usage[ ++processor_number ] != '\0' )
    {
        if ( usage[ processor_number ] == 'R' )
```

```

        return( processor_number );
    }

    return( -1 );
} /* next_receiver */

void send_processor( sender, identifier, type )
register int sender;
register char *identifier;
register char *type;
{
    register FILE *file;

    if ( ( file = fopen( file_name[ sender ], "a" ) ) == (FILE *)NULL )
    {
        fprintf( stderr, "ERROR: unable to open for update '%s'\n", file_name[ sender ] );
        exit( -1 );
    }

    fprintf( file, "** CYCLE [ %d ]\n", cycle );
    fprintf( file, "\tCALL SEND_%s(%s)\n", message_type( type ), identifier );

    fclose( file );
} /* send_processor */

void receive_processor( receiver, identifier, type )
register int receiver;
register char *identifier;
register char *type;
{
    register FILE *file;

    if ( ( file = fopen( file_name[ receiver ], "a" ) ) == (FILE *)NULL )
    {
        fprintf( stderr, "ERROR: unable to open for update '%s'\n", file_name[ receiver ] );
        exit( -1 );
    }

    fprintf( file, "** CYCLE [ %d ]\n", cycle );
    fprintf( file, "\tCALL RECEIVE_%s(%s)\n", message_type( type ), identifier );
    fclose( file );
} /* receive_processor */

void output_transfer( file, table )
register FILE *file;
register TABLE *table;
{
    register int sender;
    register int receiver;
    register char delimiter;

    if ( count( table->usage, strlen( table->usage ), 'S' ) > 1 )
    {
        fprintf( stderr, "ERROR: %s, %s, %s\n", table->identifier, table->type, table->usage );
        exit( -1 );
    }

    sender = -1;
    while ( ( sender = next_sender( sender, table->usage ) ) != -1 )
    {
        delimiter = ' ';

        receiver = -1;
        while ( ( receiver = next_receiver( receiver, table->usage ) ) != -1 )
        {
            receive_processor( receiver, table->identifier, table->type );
            fprintf( file, "%c p%02d", delimiter, receiver );

            delimiter = ',';
        }

        send_processor( sender, table->identifier, table->type );
        fprintf( file, " := p%02d.%d; [ %s %s ]\n", sender, type_length( table->type ) / 2, table->type, table->identifier );
    }
}

```

```

    table->usage = (char *)NULL;
} /* output_transfer */

int and_usage( usage1, usage2 )
register char *usage1;
register char *usage2;
{
    register int processor_number = 0;

    while ( ( usage1[ processor_number ] != '\0' ) && ( usage2[ processor_number ] != '\0' ) )
    {
        if ( ( usage1[ processor_number ] != '-' ) && ( usage2[ processor_number ] != '-' ) )
            return( 1 );

        processor_number++;
    }

    return( 0 );
} /* and_usage */

char *or_usage( usage1, usage2 )
register char *usage1;
register char *usage2;
{
    register int processor_number = 0;
    char buffer[ 256 ];

    while ( ( usage1[ processor_number ] != '\0' ) && ( usage2[ processor_number ] != '\0' ) )
    {
        if ( ( usage1[ processor_number ] == '-' ) && ( usage2[ processor_number ] != '-' ) )
            buffer[ processor_number ] = usage2[ processor_number ];
        else
            buffer[ processor_number ] = usage1[ processor_number ];

        processor_number++;
    }
    buffer[ processor_number ] = '\0';

    return( duplicate( buffer ) );
} /* or_usage */

void output_cycle( file, table_number )
register FILE *file;
register int table_number;
{
    register char *usage;

    if ( table[ table_number ].usage != (char *)NULL )
    {
        fprintf( file, "CYCLE [ %d ]\n", ++cycle );

        usage = duplicate( table[ table_number ].usage );
        output_transfer( file, &table[ table_number ] );

        while ( ++table_number != number_table )
        {
            if ( table[ table_number ].usage != (char *)NULL )
            {
                if ( and_usage( usage, table[ table_number ].usage ) == 0 )
                {
                    usage = or_usage( usage, table[ table_number ].usage );
                    output_transfer( file, &table[ table_number ] );
                }
            }

            fprintf( file, "\n" );
        }
    }
} /* output_cycle */

void output_table( file )
register FILE *file;
{

```

```

register int table_number;

fprintf( file, "LOOP\n" );
fprintf( file, "\n" );

for ( table_number = 0; table_number != number_table; table_number++ )
    output_cycle( file, table_number, number_table );
} /* output_table */

#define PROGRAM argument[ 0 ]
#define ARGUMENT argument[ argument_number ]

int main( number_argument, argument )
register int number_argument;
register char *argument[ ];
{
    register int argument_number = 0;
    int processor_number;
    char name[ 256 ];

    if ( --number_argument == 0 )
    {
        fprintf( stderr, "usage: %s 00=<file name>...31=<file name>\n", PROGRAM );
        exit( 0 );
    }

    while ( argument_number++ != number_argument )
    {
        if ( sscanf( ARGUMENT, "%d%s", &processor_number, name ) != 2 )
        {
            fprintf( stderr, "ERROR: unable to scan argument '%s'\n", ARGUMENT );
            exit( -1 );
        }

        strcpy( index( name, '.' ), ".inc" );
        file_name[ processor_number ] = duplicate( name );
        unlink( file_name[ processor_number ] );
    }

    initialize_table( stdin );
    output_table( stdout );

    exit( 0 );
} /* main */

```

FILE: usage/summary/Makefile

```

#
# Copyright 1991
# Georgia Institute of Technology
# Computer Engineering Research Laboratory
# Author: Stephen R. Wachtel
#

```

default:    summary

```

CC = cc -g
INCLUDE = include
CFLAGS = -IS(INCLUDE)
LIBRARY = library/library.a

```

```

summary.o: summary.c
    $(CC) $(CFLAGS) -c summary.c

summary:    summary.o $(LIBRARY)
    $(CC) -o summary summary.o $(LIBRARY)

```

```

clean:
    cd library; make clean
    rm -f summary summary.o

```

FILE: usage/summary/include/processor.h

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#define NUMBER_PROCESSOR 32

#define PROCESSOR struct processor_type
PROCESSOR
{
    char *file_name;
    int s;
    int r;
};

extern PROCESSOR processor[ NUMBER_PROCESSOR ];

FILE: usage/summary/include/table.h

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#define TABLE struct table_type
TABLE
{
    char *identifier;
    char *type;
    char *usage;
};

extern void allocate_table( );
extern void input_table( );
extern int compare( );
extern void sort_table( );
extern void initialize_table( );

extern TABLE *table;
extern int number_table;

FILE: usage/summary/library/Makefile

#
# Copyright 1991
# Georgia Institute of Technology
# Computer Engineering Research Laboratory
# Author: Stephen R. Wachtel
#

CC = cc -g
INCLUDE = ../include
CFLAGS = -I$(INCLUDE)
LIBRARY = library.a

OBJECTS = \
    count.o \
    duplicate.o \
    table.o

$(LIBRARY): $(OBJECTS)
    ar crv $(LIBRARY) $(OBJECTS)

```

```

ranlib $(LIBRARY)

.SUFFIXES: .c .o
.c.o:
    $(CC) -c $(CFLAGS) $<

clean:
    rm -f $(LIBRARY) $(OBJECTS)

FILE: usage/summary/library/count.c

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```

int count( string, length, c )
register char *string;
register int length;
register char c;
{
    register int c_count = 0;

    while ( length != 0 )
    {
        if ( *string == c )
            c_count++;

        string++;
        length--;
    }

    return( c_count );
} /* count */

```

FILE: usage/summary/library/duplicate.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```

#include <stdio.h>
#include <string.h>
#include <malloc.h>

```

```

char *duplicate( string )
register char *string;
{
    register char *temporary = (char *)NULL;

    if ( string != (char *)NULL )
    {
        if ( ( temporary = (char *)malloc( strlen( string ) + 1 ) ) != (char *)NULL )
            strcpy( temporary, string );
        else
            fprintf( stderr, "ERROR: duplicate( %s )\n", string );
    }

    return( temporary );
} /* duplicate */

```

FILE: usage/summary/library/table.c

```

/*
 * Copyright 1991

```

```

* Georgia Institute of Technology
* Computer Engineering Research Laboratory
* Author: Stephen R. Wachtel
*/

#include <stdio.h>
#include <malloc.h>
#include <string.h>
#include "table.h"

extern char *duplicate( );

TABLE *table = (TABLE *)NULL;
int number_table = 0;

void allocate_table( )
{
    if ( ( table = (TABLE *)malloc( sizeof( TABLE ) * number_table ) , == (TABLE *)NULL )
        {
            fprintf( stderr, "ERROR: unable to allocate 'table'\n" );
            exit( -1 );
        }
} /* allocate_table */

void input_table( file )
register FILE *file;
{
    register int table_number;
    char identifier[ 256 ];
    char type[ 256 ];
    char usage[ 256 ];

    for ( table_number = 0; table_number != number_table; table_number++ )
    {
        if ( fscanf( file, "%s %s %s\n", identifier, type, usage ) != 3 )
        {
            fprintf( stderr, "ERROR: unable to read 'table[ %d ]'\n", table_number );
            exit( -1 );
        }

        table[ table_number ].identifier = duplicate( identifier );
        table[ table_number ].type = duplicate( type );
        table[ table_number ].usage = duplicate( usage );
    }
} /* input_table */

int compare( table1, table2 )
TABLE *table1;
TABLE *table2;
{
    return( strcmp( table1->identifier, table2->identifier ) );
} /* compare */

void sort_table( )
{
    qsort( table, number_table, sizeof( TABLE ), compare );
} /* sort_table */

void initialize_table( file )
register FILE *file;
{
    fscanf( file, "%d\n", &number_table );
    allocate_table( );
    input_table( file );
    sort_table( );
} /* initialize_table */

FILE: usage/summary/summary.c

/*
* Copyright 1991

```



```

* Georgia Institute of Technology
* Computer Engineering Research Laboratory
* Author: Stephen R. Wachtel
*/

```

```

#include <stdio.h>
#include <malloc.h>
#include <string.h>
#include "table.h"
#include "processor.h"

```

```
extern char *duplicate( );
```

```
PROCESSOR processor[ NUMBER_PROCESSOR ];
```

```
void initialize_processor( )
```

```

{
    register int processor_number;

    for ( processor_number = 0; processor_number != NUMBER_PROCESSOR; processor_number++ )
    {
        processor[ processor_number ].file_name = (char *)NULL;
        processor[ processor_number ].s = 0;
        processor[ processor_number ].r = 0;
    }
} /* initialize_processor */

```

```
void output_header( file, usage )
```

```

register FILE *file;
register char *usage;
{
    register int processor_number = 0;
    static page_number = 0;

    if ( ++page_number != 1 )
        fprintf( file, "\f" );

    fprintf( file, "Page %2d", page_number );

    while ( usage[ processor_number ] != '\0' )
    {
        fprintf( file, "P%02d", processor_number );
        processor_number++;
    }

    fprintf( file, "\n" );
} /* output_header */

```

```
void output_divider( file, usage )
```

```

register FILE *file;
register char *usage;
{
    register int processor_number = 0;

    fprintf( file, "-----" );

    while ( usage[ processor_number ] != '\0' )
    {
        fprintf( file, "----" );
        processor_number++;
    }

    fprintf( file, "\n" );
} /* output_divider */

```

```
void output_transfer( file, table )
```

```

register FILE *file;
register TABLE *table;
{
    register int processor_number = 0;

    fprintf( file, "%-16s %-16s (", table->identifier, table->type );

```

```

while ( table->usage[ processor_number ] != '\0' )
{
    if ( table->usage[ processor_number ] == '-' )
        fprintf( file, "  (" );
    else
    {
        fprintf( file, " %c |", table->usage[ processor_number ] );

        switch ( table->usage[ processor_number ] )
        {
            case 'S':
                processor[ processor_number ].s++;
                break;

            case 'R':
                processor[ processor_number ].r++;
                break;

            }

        }

    processor_number++;
}

if ( count( table->usage, strlen( table->usage ), 'S' ) > 1 )
    fprintf( file, " WARNING" );

fprintf( file, "\n" );
} /* output_transfer */

void output_table( FILE *file )
register FILE *file;
{
#define LINE_NUMBER 64
    register int table_number;
    register int line_number = LINE_NUMBER;

    for ( table_number = 0; table_number != number_table; table_number++ )
    {
        if ( line_number == LINE_NUMBER )
        {
            line_number = 0;

            output_header( file, table[ table_number ].usage );
            line_number++;
            output_divider( file, table[ table_number ].usage );
            line_number++;
        }

        output_transfer( file, &table[ table_number ] );
        line_number++;
        output_divider( file, table[ table_number ].usage );
        line_number++;
    }
} /* output_table */

void output_processor( FILE *file )
register FILE *file;
{
    register int processor_number;

    fprintf( file, "\n" );

    for ( processor_number = 0; processor_number != NUMBER_PROCESSOR; processor_number++ )
    {
        if ( processor[ processor_number ].file_name != (char *)NULL )
        {
            strcpy( index( processor[ processor_number ].file_name, '.' ), ".for" );
            fprintf( file, "%02d = %s", processor_number, processor[ processor_number ].file_name );

            fprintf( file, ", S = %3d", processor[ processor_number ].s );
            fprintf( file, ", R = %3d", processor[ processor_number ].r );
            fprintf( file, ", %3d", processor[ processor_number ].s + processor[ processor_number ].r );

            fprintf( file, "\n" );
        }
    }
}

```

```

) /* output_processor */

#define PROGRAM argument[ 0 ]
#define ARGUMENT argument[ argument_number ]

int main( number_argument, argument )
register int number_argument;
register char *argument[ ];
{
    register int argument_number = 0;
    int processor_number;
    char file_name[ 256 ];

    if ( --number_argument == 0 )
    {
        fprintf( stderr, "usage: %s 00=<file name>...31=<file name>\n", PROGRAM );
        exit( 0 );
    }

    initialize_processor( );

    while ( argument_number++ != number_argument )
    {
        if ( sscanf( ARGUMENT, "%d=%s", &processor_number, file_name ) != 2 )
        {
            fprintf( stderr, "ERROR: unable to scan argument '%s'\n", ARGUMENT );
            exit( -1 );
        }

        processor[ processor_number ].file_name = duplicate( file_name );
    }

    initialize_table( stdin );
    output_table( stdout );
    output_processor( stdout );

    exit( 0 );
} /* main */

```

FILE: usage/type/Makefile

```

#
# Copyright 1991
# Georgia Institute of Technology
# Computer Engineering Research Laboratory
# Author: Stephen R. Wachtel
#

default:    type

CC = cc -g
INCLUDE = include
CFLAGS = -I$(INCLUDE)
LIBRARY = statement/statement.a library/library.a

OBJECTS = \
    $(INCLUDE)/grammar.h \
    *grammar.[co] \
    *scanner.[co] \
    yytrace.[co] \
    y.output

PROGRAMS = \
    *type

grammar.c: grammar.y
    yacc -dv grammar.y
    mv y.tab.h $(INCLUDE)/grammar.h
    mv y.tab.c grammar.c

scanner.c: scanner.l
    lex -vt scanner.l | sed 's/getc/yygetc/' >scanner.c

```

```

scanner.o: scanner.c $(INCLUDE)/grammar.h
    $(CC) $(CFLAGS) -c scanner.c

grammar.o: grammar.c
    $(CC) $(CFLAGS) -c grammar.c

type: grammar.o scanner.o $(LIBRARY)
    $(CC) -o type grammar.o scanner.o $(LIBRARY)

sgrammar.c: grammar.c yytoken.awk
    awk -f yytoken.awk <grammar.c >sgrammar.c

sgrammar.o: sgrammar.c
    $(CC) $(CFLAGS) -c sgrammar.c

stype: sgrammar.o scanner.o $(LIBRARY)
    $(CC) -o stype sgrammar.o scanner.o $(LIBRARY)

dscanner.c: scanner.c
    cp scanner.c dscanner.c

dscanner.o: dscanner.c $(INCLUDE)/grammar.h
    $(CC) $(CFLAGS) -DDEBUG -c dscanner.c

dtype: grammar.o dscanner.o $(LIBRARY)
    $(CC) -o dtype grammar.o dscanner.o $(LIBRARY)

tgrammar.c: grammar.c
    sed 's/yystack:/& yytrace(yystate);/' <grammar.c >tgrammar.c

tgrammar.o: tgrammar.c
    $(CC) $(CFLAGS) -c tgrammar.c

ttype: tgrammar.o scanner.o yytrace.o $(LIBRARY)
    $(CC) -o ttype tgrammar.o scanner.o yytrace.o $(LIBRARY)

yytrace.c: grammar.c yytrace.awk
    awk -f yytrace.awk <y.output >yytrace.c

yytrace.o: yytrace.c
    $(CC) $(CFLAGS) -c yytrace.c

clean:
    cd statement; make clean
    cd library; make clean
    rm -f $(PROGRAMS) $(OBJECTS)

FILE: usage/type/grammar.y

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

/*
 * FORTRAN 77
 */

%token RW_AND
%token RW_ASSIGN
%token RW_BACKSPACE
%token RW_BLOCK_DATA
%token RW_CALL
%token RW_CHARACTER
%token RW_CLOSE
%token RW_COMMON
%token RW_COMPLEX
%token RW_CONTINUE

```

```

%token RW_DATA
%token RW_DIMENSION
%token RW_DO
%token RW_DOUBLE_PRECISION
%token RW_ELSE
%token RW_ELSE_IF
%token RW_END
%token RW_END IF
%token RW_ENDFILE
%token RW_ENTRY
%token RW_EQ
%token RW_EQUIVALENCE
%token RW_EQV
%token RW_EXTERNAL
%token RW_FALSE
%token RW_FORMAT
%token RW_FUNCTION
%token RW_GE
%token RW_GO_TO
%token RW_GT
%token RW_IF
%token RW_IMPLICIT
%token RW_INCLUDE
%token RW_INQUIRE
%token RW_INTEGER
%token RW_INTRINSIC
%token RW_LE
%token RW_LOGICAL
%token RW_LT
%token RW_NAMELIST
%token RW_NE
%token RW_NEQV
%token RW_NOT
%token RW_OPEN
%token RW_OR
%token RW_PARAMETER
%token RW_PAUSE
%token RW_PRINT
%token RW_PROGRAM
%token RW_READ
%token RW_REAL
%token RW_RETURN
%token RW_REWIND
%token RW_SAVE
%token RW_STOP
%token RW_SUBROUTINE
%token RW_THEN
%token RW_TO
%token RW_TRUE
%token RW_WRITE
%token RW_UNDEFINED

```

```

%token COMMENT
%token CONCATENATE
%token DOUBLE_PRECISION
%token EXPONENTIATE
%token HOLLERITH
%token IDENTIFIER
%token INTEGER
%token LABEL
%token REAL
%token STRING

```

```

%left ','
%nonassoc ':'
%right '='
%left RW_EQV RW_NEQV
%left RW_OR
%left RW_AND
%left RW_NOT
%nonassoc RW_EQ RW_NE RW_LT RW_LE RW_GT RW_GE
%left CONCATENATE
%left '+' '-'
%left '*' '/'
%right EXPONENTIATE
%left SIGN

```

```

%{

```

```

typedef char *POINTER;
#define YYSTYPE POINTER

extern POINTER duplicate( );
extern POINTER merge( );
extern POINTER type( );

#include "list.h"
#include "class.h"
%)

%%

program:
    optional_statement_list
    {
        summary( );
    }
    ;

optional_statement_list:
    /* NULL */
    |
    statement_list
    ;

statement_list:
    statement
    |
    statement_list statement
    ;

statement:
    comment_statement
    |
    label unlabeled_statement
    ;

comment_statement:
    COMMENT
    ;

label:
    LABEL
    ;

unlabeled_statement:
    include_statement
    |
    program_statement
    |
    block_data_statement
    |
    function_statement
    |
    subroutine_statement
    |
    entry_statement
    |
    end_statement
    |
    specification_statement
    |
    executable_statement
    |
    format_statement
    ;

include_statement:
    RW_INCLUDE character_constant
    ;

```

```

program_statement:
    RW_PROGRAM program_identifier
    ;

program_identifier:
    IDENTIFIER
    ;

block_data_statement:
    RW_BLOCK_DATA block_data_identifier
    ;

block_data_identifier:
    IDENTIFIER
    ;

function_statement:
    RW_FUNCTION function_identifier optional_formal_argument_list
    {
        function_statement( 0, $2, $3 );
    }
    |
    type RW_FUNCTION function_identifier optional_formal_argument_list
    {
        function_statement( $1, $3, $4 );
    }
    ;

function_identifier:
    IDENTIFIER
    ;

subroutine_statement:
    RW_SUBROUTINE subroutine_identifier
    |
    RW_SUBROUTINE subroutine_identifier optional_formal_argument_list
    ;

subroutine_identifier:
    IDENTIFIER
    ;

entry_statement:
    RW_ENTRY entry_identifier
    |
    RW_ENTRY entry_identifier optional_formal_argument_list
    ;

entry_identifier:
    IDENTIFIER
    {
        $$ = $1;
    }
    ;

optional_formal_argument_list:
    '(' ')'
    {
        $$ = 0;
    }
    |
    '(' formal_argument_list ')'
    {
        $$ = $2;
    }
    ;

formal_argument_list:
    formal_argument

```

```

        {
            $$ = merge( "%s)", $1 );
        }
        |
        formal_argument_list ',' formal_argument
        {
            $$ = merge( "%s%s)", $1, $3 );
        }
    ;

formal_argument:
    IDENTIFIER
    {
        $$ = $1;
    }
    |
    formal_argument_altername_return
    {
        $$ = $1;
    }
    ;

formal_argument_altername_return:
    '*'
    {
        $$ = duplicate( "*" );
    }
    ;

end_statement:
    RW_END
    {
        end_statement( );
    }
    ;

specification_statement:
    external_statement
    |
    intrinsic_statement
    |
    parameter_statement
    |
    dimension_statement
    |
    declaration_statement
    |
    save_statement
    |
    common_statement
    |
    equivalence_statement
    |
    implicit_statement
    |
    data_statement
    |
    namelist_statement
    ;

external_statement:
    RW_EXTERNAL external_list
    ;

external_list:
    external
    {
        $$ = merge( "%s)", $1 );
    }
    |
    external_list ',' external
    {
        $$ = merge( "%s%s)", $1, $3 );
    }
    ;

```



```

external:
    IDENTIFIER
    {
        $$ = $1;
    }
    ;

intrinsic_statement:
    RW_INTRINSIC intrinsic_list
    ;

intrinsic_list:
    intrinsic
    {
        $$ = merge( "${s}", $1 );
    }
    |
    intrinsic_list ',' intrinsic
    {
        $$ = merge( "%s${s}", $1, $3 );
    }
    ;

intrinsic:
    IDENTIFIER
    {
        $$ = $1;
    }
    ;

parameter_statement:
    RW_PARAMETER '(' parameter_list ')'
    {
        parameter_statement( $3 );
    }
    ;

parameter_list:
    parameter
    {
        $$ = merge( "${s}", $1 );
    }
    |
    parameter_list ',' parameter
    {
        $$ = merge( "%s${s}", $1, $3 );
    }
    ;

parameter:
    IDENTIFIER '=' expression
    {
        $$ = merge( "${s}${s}", $1, $3 );
    }
    ;

dimension_statement:
    RW_DIMENSION dimension_list
    {
        dimension_statement( $2 );
    }
    ;

dimension_list:
    dimension
    {
        $$ = merge( "${s}", $1 );
    }
    |
    dimension_list ',' dimension
    {

```

```

        $$ = merge( "%s{%s}", $1, $3 );
    }
;

dimension:
    IDENTIFIER '(' subscript_list ')'
    {
        $$ = merge( "%s{%s}", $1, $3 );
    }
;

subscript_list:
    subscript
    {
        $$ = merge( "%s", $1 );
    }
    |
    subscript_list ',' subscript
    {
        $$ = merge( "%s{%s}", $1, $3 );
    }
;

subscript:
    upper_bound
    {
#define LOWER_BOUND duplicate( "1" )
        $$ = merge( "%s{%s}", LOWER_BOUND, $1 );
    }
    |
    lower_bound ':' upper_bound
    {
        $$ = merge( "%s{%s}", $1, $3 );
    }
;

lower_bound:
    expression
    {
        $$ = $1;
    }
;

upper_bound:
    lower_bound
    {
        $$ = $1;
    }
    |
    upper_bound_adjustable
    {
        $$ = $1;
    }
;

upper_bound_adjustable:
    '*'
    {
        $$ = duplicate( "*" );
    }
;

declaration_statement:
    type declaration_list
    {
        declaration_statement( $1, $2 );
    }
;

declaration_list:
    declaration
    {
        $$ = merge( "%s", $1 );
    }

```

```

    }
    |
    declaration_list ',' declaration
    {
        $$ = merge( "%s{%s}", $1, $3 );
    }
    ;

declaration:
    IDENTIFIER
    {
        $$ = merge( "%{s}", $1 );
    }
    |
    IDENTIFIER '(' subscript_list ')'
    {
        $$ = merge( "%{s}{%s}", $1, $3 );
    }
    ;

type:
    type_name optional_type_length
    {
        $$ = type( $1, $2 );
    }
    ;

type_name:
    RW_CHARACTER
    {
        $$ = duplicate( "CHARACTER" );
    }
    |
    RW_COMPLEX
    {
        $$ = duplicate( "COMPLEX" );
    }
    |
    RW_DOUBLE_PRECISION
    {
        $$ = duplicate( "DOUBLE_PRECISION" );
    }
    |
    RW_INTEGER
    {
        $$ = duplicate( "INTEGER" );
    }
    |
    RW_LOGICAL
    {
        $$ = duplicate( "LOGICAL" );
    }
    |
    RW_REAL
    {
        $$ = duplicate( "REAL" );
    }
    |
    RW_UNDEFINED
    {
        $$ = duplicate( "UNDEFINED" );
    }
    ;

optional_type_length:
    /* NULL */
    {
        $$ = 0;
    }
    |
    type_length
    {
        $$ = $1;
    }
    ;

```

```

type_length:
  '*' INTEGER
  {
    $$ = $2;
  }
  |
  '*' type_length_adjustable
  {
    $$ = $2;
  }
  ;

type_length_adjustable:
  '(' '*' ')'
  {
    $$ = duplicate( "-1" );
  }
  ;

save_statement:
  RW_SAVE optional_save_list
  ;

optional_save_list:
  /* NULL */
  {
    $$ = 0;
  }
  |
  save_list
  {
    $$ = $1;
  }
  ;

save_list:
  save
  {
    $$ = merge( "%s", $1 );
  }
  |
  save_list ',' save
  {
    $$ = merge( "%s%s", $1, $3 );
  }
  ;

save:
  IDENTIFIER
  {
    $$ = $1;
  }
  |
  common_name
  {
    $$ = $1;
  }
  ;

common_statement:
  RW_COMMON optional_common_name common_variable_list
  {
    common_statement( $2, $3 );
  }
  ;

optional_common_name:
  /* NULL */
  {
    $$ = 0;
  }
  |
  common_name
  {

```

```

        $$ = $1;
    }
;

common_name:
    '/' optional_identifier '/'
    {
        $$ = $2;
    }
;

optional_identifier:
    /* NULL */
    {
        $$ = 0;
    }
    |
    IDENTIFIER
    {
        $$ = $1;
    }
;

common_variable_list:
    common_variable
    {
        $$ = merge( "${s}", $1 );
    }
    |
    common_variable_list ',' common_variable
    {
        $$ = merge( "${s}${s}", $1, $3 );
    }
;

common_variable:
    IDENTIFIER
    {
        $$ = merge( "${s}", $1 );
    }
    |
    IDENTIFIER '(' subscript_list ')'
    {
        $$ = merge( "${s}${s}", $1, $3 );
    }
;

equivalence_statement:
    RW_EQUIVALENCE equivalence_list
;

equivalence_list:
    equivalence
    {
        $$ = merge( "${s}", $1 );
    }
    |
    equivalence_list ',' equivalence
    {
        $$ = merge( "${s}${s}", $1, $3 );
    }
;

equivalence:
    '(' equivalence_variable_list ')'
    {
        $$ = $2;
    }
;

equivalence_variable_list:
    equivalence_variable
    {

```

```

        $$ = merge( "{%s}", $1 );
    }
    |
    equivalence_variable_list ',' equivalence_variable
    {
        $$ = merge( "%s{%s}", $1, $3 );
    }
    ;

```

```

equivalence_variable:
    IDENTIFIER
    {
        $$ = merge( "{%s}", $1 );
    }
    |
    IDENTIFIER '(' subscript_list ')'
    {
        $$ = merge( "{%s}{%s}", $1, $3 );
    }
    ;

```

```

implicit_statement:
    RW_IMPLICIT type '(' implicit_list ')'
    {
        implicit_statement( $2, $4 );
    }
    ;

```

```

implicit_list:
    implicit
    {
        $$ = merge( "{%s}", $1 );
    }
    |
    implicit_list ',' implicit
    {
        $$ = merge( "%s{%s}", $1, $3 );
    }
    ;

```

```

implicit:
    IDENTIFIER
    {
        $$ = merge( "{%s}", $1 );
    }
    |
    IDENTIFIER '-' IDENTIFIER
    {
        $$ = merge( "{%s}{%s}", $1, $3 );
    }
    ;

```

```

namelist_statement:
    RW_NAMELIST namelist_name namelist_list
    ;

```

```

namelist_name:
    '/' IDENTIFIER '/'
    {
        $$ = $2;
    }
    ;

```

```

namelist_list:
    namelist
    {
        $$ = merge( "{%s}", $1 );
    }
    |
    namelist_list ',' namelist
    {
        $$ = merge( "%s{%s}", $1, $3 );
    }
    ;

```

```

namelist:
  IDENTIFIER
  {
    $$ = $1;
  }
;

data_statement:
  RW_DATA data_list
;

data_list:
  data
  {
    $$ = merge( "%s", $1 );
  }
|
  data_list optional_comma data
  {
    $$ = merge( "%s%s", $1, $3 );
  }
;

data:
  data_variable_list '/' data_constant_list '/'
  {
    $$ = merge( "%s{%s}", $1, $3 );
  }
;

data_variable_list:
  data_variable
  {
    $$ = merge( "%s", $1 );
  }
|
  data_variable_list ',' data_variable
  {
    $$ = merge( "%s{%s}", $1 $3 );
  }
;

data_variable:
  variable
  {
    $$ = $1;
  }
|
  data_implied_do_list
  {
    $$ = $1;
  }
;

data_implied_do_list:
  '(' data_variable_list ',' IDENTIFIER '=' expression_list ')'
  {
    add_list( 0, $4, 0, 0, IMPLICIT | LOCAL | VARIABLE );
    $$ = merge( "( %s, %s = %s )", list( $2, ",", " ), $4, list( $6, ",", " ) );
  }
;

data_constant_list:
  data_constant
  {
    $$ = merge( "%s", $1 );
  }
|
  data_constant_list ',' data_constant
  {
    $$ = merge( "%s{%s}", $1, $3 );
  }
;

```

```

;

data_constant:
  data_initialization
  {
    $$ = $1;
  }
  |
  IDENTIFIER '*' data_initialization
  {
    $$ = merge( "%s * %s", $1, $3 );
  }
  |
  INTEGER '*' data_initialization
  {
    $$ = merge( "%s * %s", $1, $3 );
  }
;

data_initialization:
  IDENTIFIER
  {
    $$ = $1;
  }
  |
  character_constant
  {
    $$ = $1;
  }
  |
  logical_constant
  {
    $$ = $1;
  }
  |
  signed_numerical_constant
  {
    $$ = $1;
  }
;

signed_numerical_constant:
  numerical_constant
  {
    $$ = $1;
  }
  |
  '+' numerical_constant %prec SIGN
  {
    $$ = merge( "+%s", $2 );
  }
  |
  '-' numerical_constant %prec SIGN
  {
    $$ = merge( "-%s", $2 );
  }
;

expression:
  parenthesis_expression
  {
    $$ = $1;
  }
  |
  simple_expression
  {
    $$ = $1;
  }
;

parenthesis_expression:
  '(' expression ')'
  {
    $$ = merge( "( %s )", $2 );
  }
;

```



```

simple_expression:
  variable
  {
    $$ = $1;
  }
  |
  constant
  {
    $$ = $1;
  }
  |
  arithmetic_expression
  {
    $$ = $1;
  }
  |
  character_expression
  {
    $$ = $1;
  }
  |
  relational_expression
  {
    $$ = $1;
  }
  |
  logical_expression
  {
    $$ = $1;
  }
  |
  unary_expression
  {
    $$ = $1;
  }
  ;

variable:
  IDENTIFIER
  {
    add_list( 0, $1, 0, 0, IMPLICIT | LOCAL | VARIABLE );
    $$ = $1;
  }
  |
  IDENTIFIER string_subset
  {
    add_list( 0, $1, 0, 0, IMPLICIT | LOCAL | VARIABLE );
    $$ = merge( "%s%s", $1, $2 );
  }
  |
  array
  {
    $$ = $1;
  }
  ;

array:
  IDENTIFIER '(' optional_expression_list ')'
  {
    if ( !array( $1 ) )
    {
      add_list( 0, $1, 0, 0, IMPLICIT | GLOBAL | VARIABLE | FUNCTION );
    }
    $$ = merge( "%s( %s )", $1, list( $3, " ", " ) );
  }
  |
  IDENTIFIER '(' optional_expression_list ')' string_subset
  {
    if ( !array( $1 ) )
    {
      add_list( 0, $1, 0, 0, IMPLICIT | GLOBAL | VARIABLE | FUNCTION );
    }
    $$ = merge( "%s( %s )%s", $1, list( $3, " ", " ), $5 );
  }
  ;

```

```

optional_expression_list:
/* NULL */
{
    $$ = 0;
}
|
expression_list
{
    $$ = $1;
}
;

expression_list:
expression
{
    $$ = merge( "%s", $1 );
}
|
expression_list ',' expression
{
    $$ = merge( "%s%s", $1, $3 );
}
;

string_subset:
'(' optional_expression ':' optional_expression ')'
{
    $$ = merge( "( %s : %s )", $2, $4 );
}
;

optional_expression:
/* NULL */
{
    $$ = 0;
}
|
expression
{
    $$ = $1;
}
;

constant:
character_constant
{
    $$ = $1;
}
|
logical_constant
{
    $$ = $1;
}
|
numerical_constant
{
    $$ = $1;
}
;

character_constant:
HOLLERITH
{
    $$ = $1;
}
|
STRING
{
    $$ = $1;
}
;

logical_constant:
RW_FALSE
{

```

```

    $$ = duplicate( ".FALSE." );
}
|
RW_TRUE
{
    $$ = duplicate( ".TRUE." );
}
;

numerical_constant:
DOUBLE_PRECISION
{
    $$ = $1;
}
|
INTEGER
{
    $$ = $1;
}
|
REAL
{
    $$ = $1;
}
;

arithmetic_expression:
expression '+' expression %prec '+'
{
    $$ = merge( "%s + %s", $1, $3 );
}
|
expression '-' expression %prec '-'
{
    $$ = merge( "%s - %s", $1, $3 );
}
|
expression '*' expression %prec '*'
{
    $$ = merge( "%s * %s", $1, $3 );
}
|
expression '/' expression %prec '/'
{
    $$ = merge( "%s / %s", $1, $3 );
}
|
expression EXPONENTIATE expression %prec EXPONENTIATE
{
    $$ = merge( "%s ** %s", $1, $3 );
}
;

character_expression:
expression '/' '/' expression %prec CONCATENATE
{
    $$ = merge( "%s // %s", $1, $4 );
}
;

relational_expression:
expression RW_EQ expression %prec RW_EQ
{
    $$ = merge( "%s .EQ. %s", $1, $3 );
}
|
expression RW_NE expression %prec RW_NE
{
    $$ = merge( "%s .NE. %s", $1, $3 );
}
|
expression RW_LT expression %prec RW_LT
{
    $$ = merge( "%s .LT. %s", $1, $3 );
}
|
expression RW_LE expression %prec RW_LE

```

```

    {
        $$ = merge( "%s .LE. %s", $1, $3 );
    }
    |
    expression RW_GT expression %prec RW_GT
    {
        $$ = merge( "%s .GT. %s", $1, $3 );
    }
    |
    expression RW_GE expression %prec RW_GE
    {
        $$ = merge( "%s .GE. %s", $1, $3 );
    }
    ;

logical_expression:
    expression RW_AND expression %prec RW_AND
    {
        $$ = merge( "%s .AND. %s", $1, $3 );
    }
    |
    expression RW_OR expression %prec RW_OR
    {
        $$ = merge( "%s .OR. %s", $1, $3 );
    }
    |
    expression RW_EQV expression %prec RW_EQV
    {
        $$ = merge( "%s .EQV. %s", $1, $3 );
    }
    |
    expression RW_NEQV expression %prec RW_NEQV
    {
        $$ = merge( "%s .NEQV. %s", $1, $3 );
    }
    ;

unary_expression:
    '+' expression %prec SIGN
    {
        $$ = merge( "+%s", $2 );
    }
    |
    '-' expression %prec SIGN
    {
        $$ = merge( "-%s", $2 );
    }
    |
    RW_NOT expression %prec RW_NOT
    {
        $$ = merge( ".NOT. %s", $2 );
    }
    ;

executable_statement:
    do_statement
    |
    logical_if_statement
    |
    block_if_statement
    |
    else_statement
    |
    else_if_statement
    |
    end_if_statement
    |
    subset_executable_statement
    ;

do_statement:
    RW_DO optional_integer IDENTIFIER '=' expression_list
    {
        do_statement( $2, $3, $5 );
    }
    ;

```

```

optional_integer:
    /* NULL */
    {
        $$ = 0;
    }
    |
    INTEGER
    {
        $$ = $1;
    }
    ;

logical_if_statement:
    if_expression subset_executable_statement
    ;

if_expression:
    RW_IF '(' expression ')'
    ;

block_if_statement:
    RW_IF '(' expression ')' RW_THEN
    ;

else_statement:
    RW_ELSE
    ;

else_if_statement:
    RW_ELSE_IF '(' expression ')' RW_THEN
    ;

end_if_statement:
    RW_END_IF
    ;

subset_executable_statement:
    assignment_statement
    |
    assign_statement
    |
    arithmetic_if_statement
    |
    continue_statement
    |
    call_statement
    |
    return_statement
    |
    unconditional_go_to_statement
    |
    computed_go_to_statement
    |
    assigned_go_to_statement
    |
    stop_statement
    |
    pause_statement
    |
    io_statement
    ;

assignment_statement:
    variable '=' expression
    ;

assign_statement:
    RW_ASSIGN INTEGER RW_TO IDENTIFIER
    ;

```

```

arithmetic_if_statement:
    RW_IF '(' expression ')' integer_list
    ;

continue_statement:
    RW_CONTINUE
    ;

call_statement:
    RW_CALL IDENTIFIER
    | RW_CALL IDENTIFIER optional_actual_argument_list
    ;

optional_actual_argument_list:
    '(' ')'
    {
        $$ = 0;
    }
    | '(' actual_argument_list ')'
    {
        $$ = $2;
    }
    ;

actual_argument_list:
    actual_argument
    {
        $$ = merge( "%s", $1 );
    }
    | actual_argument_list ',' actual_argument
    {
        $$ = merge( "%s%s", $1, $3 );
    }
    ;

actual_argument:
    expression
    {
        $$ = $1;
    }
    | actual_argument_alternate_return
    {
        $$ = $1;
    }
    ;

actual_argument_alternate_return:
    '*' INTEGER
    {
        $$ = merge( "%*s", $2 );
    }
    ;

return_statement:
    RW_RETURN optional_expression
    ;

unconditional_go_to_statement:
    RW_GO_TO INTEGER
    ;

computed_go_to_statement:
    RW_GO_TO '(' integer_list ')' optional_comma expression
    ;

assigned_go_to_statement:
    RW_GO_TO IDENTIFIER

```

```

|
|  RW_GO_TO IDENTIFIER optional_comma '(' integer_list ')'
|
;

optional_comma:
/* NULL */
|
|  ','
|
;

integer_list:
INTEGER
{
    $$ = merge( "%s", $1 );
}
|
integer_list ',' INTEGER
{
    $$ = merge( "%s%s", $1, $3 );
}
;

pause_statement:
RW_PAUSE optional_expression
;

stop_statement:
RW_STOP optional_expression
;

io_statement:
open_statement
|
close_statement
|
inquire_statement
|
read_statement
|
write_statement
|
print_statement
|
backspace_statement
|
rewind_statement
|
endfile_statement
;

open_statement:
RW_OPEN '(' control_information_list ')'
;

close_statement:
RW_CLOSE '(' control_information_list ')'
;

inquire_statement:
RW_INQUIRE '(' control_information_list ')'
;

read_statement:
RW_READ '(' control_information_list ')' optional_io_list
|
RW_READ control
|
RW_READ control ',' io_list
;

write_statement:

```

```

        RW_WRITE '(' control_information_list ')' optional_io_list
;

print_statement:
    RW_PRINT control
    |
    RW_PRINT control ',' io_list
;

backspace_statement:
    RW_BACKSPACE '(' control_information_list ')'
    |
    RW_BACKSPACE control
;

rewind_statement:
    RW_REWIND '(' control_information_list ')'
    |
    RW_REWIND control
;

endfile_statement:
    RW_ENDFILE '(' control_information_list ')'
    |
    RW_ENDFILE control
;

control_information_list:
    control_information
    {
        $$ = merge( "%s)", $1 );
    }
    |
    control_information_list ',' control_information
    {
        $$ = merge( "%s%s)", $1, $3 );
    }
;

control_information:
    control
    {
        $$ = $1;
    }
    |
    IDENTIFIER '=' expression
    {
        $$ = merge( "%s = %s", $1, $3 );
    }
;

control:
    variable
    {
        $$ = $1;
    }
    |
    constant
    {
        $$ = $1;
    }
    |
    '*'
    {
        $$ = duplicate( "*" );
    }
;

optional_io_list:
    /* NULL */
    {
        $$ = 0;
    }

```



```

    |
    io_list
    {
        $$ = $1;
    }
;

io_list:
    io
    {
        $$ = merge( "${s}", $1 );
    }
    |
    io_list ',' io
    {
        $$ = merge( "${s}${s}", $1, $3 );
    }
;

io:
    expression
    {
        $$ = $1;
    }
    |
    io_implied_do_list
    {
        $$ = $1;
    }
;

io_implied_do_list:
    '(' io_list ',' IDENTIFIER '=' expression_list ')'
    {
        add_list( 0, $4, 0, 0, IMPLICIT | LOCAL | VARIABLE );
        $$ = merge( "( ${s}, ${s} = ${s} )", list( $2, " ", " ), $4, list( $6, " ", " ) );
    }
;

format_statement:
    RW_FORMAT
;

%%

FILE: usage/type/include/class.h

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#define IMPLICIT 0x00
#define EXPLICIT 0x01

#define LOCAL 0x00
#define GLOBAL 0x02

#define VARIABLE 0x00
#define CONSTANT 0x04

#define ARRAY 0x10
#define FUNCTION 0x20

FILE: usage/type/include/list.h

/*
 * Copyright 1991
 * Georgia Institute of Technology

```

```

* Computer Engineering Research Laboratory
* Author: Stephen R. Wachtel
*/

```

```

#define LIST struct list_type
LIST
{
    char *structure;
    char *identifier;
    char *type;
    char *list;
    int class;
    LIST *next;
};

```

```

extern LIST *end_list( );
extern LIST *add_end_list( );
extern LIST *find_list( );
extern void add_list( );
extern int length_list( );
extern void print_class( );
extern void print_list( );

```

```

extern LIST *_list_;

```

```

FILE: usage/type/library/Makefile

```

```

#
# Copyright 1991
# Georgia Institute of Technology
# Computer Engineering Research Laboratory
# Author: Stephen R. Wachtel
#

```

```

CC = cc -g
INCLUDE = ../include
CFLAGS = -I$(INCLUDE)
LIBRARY = library.a

```

```

OBJECTS = \
    array.o \
    count.o \
    duplicate.o \
    hollerith.o \
    implicit.o \
    link_list.o \
    list.o \
    main.o \
    merge.o \
    non_blank.o \
    parse.o \
    summary.o \
    type.o \
    uppercase.o \
    yyerror.o \
    yygetc.o \
    yywrap.o

```

```

$(LIBRARY): $(OBJECTS)
    ar crv $(LIBRARY) $(OBJECTS)
    ranlib $(LIBRARY)

```

```

.SUFFIXES: .c .o
.c.o:
    $(CC) -c $(CFLAGS) $<

```

```

clean:
    rm -f $(LIBRARY) $(OBJECTS)

```

```

FILE: usage/type/library/array.c

```

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```

#include <stdio.h>
#include "list.h"
#include "class.h"

```

```

int array( identifier )
register char *identifier;
{
    register LIST *temporary = find_list( _list_, identifier );

    if ( temporary != (LIST *)NULL )
        return( ( temporary->class & ARRAY ) == ARRAY );
    else
        return( 0 );
} /* array */

```

FILE: usage/type/library/count.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```

int count( string, length, c )
register char *string;
register int length;
register char c;
{
    register int c_count = 0;

    while ( length != 0 )
    {
        if ( *string == c )
            c_count++;

        string++;
        length--;
    }

    return( c_count );
} /* count */

```

FILE: usage/type/library/duplicate.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```

#include <stdio.h>
#include <string.h>
#include <malloc.h>

```

```

char *duplicate( string )
register char *string;
{
    register char *temporary = (char *)NULL;

    if ( string != (char *)NULL )
    {

```

```

        if ( ( temporary = (char *)malloc( strlen( string ) + 1 ) ) != (char *)NULL )
            strcpy( temporary, string );
        else
            fprintf( stderr, "ERROR: duplicate( %s )\n", string );
    }

    return( temporary );
} /* duplicate */

```

FILE: usage/type/library/hollerith.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include <stdio.h>

char *hollerith( string, delimiter )
register char *string;
register char delimiter;
{
    int hollerith_length;
    register int string_length = 0;

    sscanf( string, "%dh", &hollerith_length );

    string[ string_length++ ] = delimiter;
    while ( hollerith_length != 0 )
    {
        if ( ( string[ string_length ] = yyinput( ) ) == '\n' )
        {
            yyunput( string[ string_length ] );
            break;
        }

        string_length++;
        hollerith_length--;
    }
    string[ string_length++ ] = delimiter;

    string[ string_length ] = '\0';
    return( string );
} /* hollerith */

```

FILE: usage/type/library/implicit.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

extern char *default_integer;
extern char *default_logical;
extern char *duplicate( );

static char *implicit_table[ ] =
{
    /* A */ 0,
    /* B */ 0,
    /* C */ 0,
    /* D */ 0,
    /* E */ 0,
    /* F */ 0,
    /* G */ 0,
    /* H */ 0,
    /* I */ 0,
    /* J */ 0,
    /* K */ 0,

```

```

/* L */ 0,
/* M */ 0,
/* N */ 0,
/* O */ 0,
/* P */ 0,
/* Q */ 0,
/* R */ 0,
/* S */ 0,
/* T */ 0,
/* U */ 0,
/* V */ 0,
/* W */ 0,
/* X */ 0,
/* Y */ 0,
/* Z */ 0,

/* ? */ "UNDEFINED"
};

#define IMPLICIT_TABLE ( sizeof( implicit_table ) / sizeof( char * ) )

int offset( c )
register char *c;
{
#define LOWER_CASE( c ) ( ( c >= 'a' ) && ( c <= 'z' ) )
    if ( LOWER_CASE( c[ 0 ] ) )
        return( c[ 0 ] - 'a' );

#define UPPER_CASE( c ) ( ( c >= 'A' ) && ( c <= 'Z' ) )
    if ( UPPER_CASE( c[ 0 ] ) )
        return( c[ 0 ] - 'A' );

    return( IMPLICIT_TABLE - 1 );
} /* offset */

char *implicit_type( string )
register char *string;
{
    return( duplicate( implicit_table[ offset( string ) ] ) );
} /* implicit_type */

void type_implicit( string, lower_bound, upper_bound )
register Char *string;
register char *lower_bound;
register char *upper_bound;
{
    register int index;

    if ( upper_bound == 0 )
        upper_bound = lower_bound;

    for ( index = offset( lower_bound ); index <= offset( upper_bound ); index++ )
        implicit_table[ index ] = string;
} /* type_implicit */

void implicit_initialize( )
{
    type_implicit( "REAL*4", "A", "H" );
    type_implicit( default_integer, "I", "N" );
    type_implicit( "REAL*4", "C", "Z" );
} /* implicit_initialize */

FILE: usage/type/library/link_list.c

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include <stdio.h>
#include <malloc.h>

```

```

#include "list.h"
#include "class.h"

extern char *duplicate( );
extern char *implicit_type( );

LIST *_list = (LIST *)NULL;
#define ZERO( a, b ) ( ( a != 0 ) ? a : b )

LIST *end_list( list )
register LIST *list;
{
    if ( list != (LIST *)NULL )
    {
        while ( list->next != (LIST *)NULL )
            list = list->next;
    }

    return( list );
} /* end_list */

LIST *add_end_list( list, identifier )
register LIST **list;
register char *identifier;
{
    register LIST *temporary = (LIST *)malloc( sizeof( LIST ) );

    temporary->structure = (char *)NULL;
    temporary->identifier = identifier;
    temporary->type = (char *)NULL;
    temporary->list = (char *)NULL;
    temporary->class = 0;
    temporary->next = (LIST *)NULL;

    if ( *list == (LIST *)NULL )
        *list = temporary;
    else
        end_list( *list )->next = temporary;

    return( temporary );
} /* add_end_list */

LIST *find_list( list, identifier )
register LIST *list;
register char *identifier;
{
    while ( list != (LIST *)NULL )
    {
        if ( strcmp( list->identifier, identifier ) == 0 )
            return( list );

        list = list->next;
    }

    return( (LIST *)NULL );
} /* find_list */

void add_list( structure, identifier, type, list, class )
register char *structure;
register char *identifier;
register char *type;
register char *list;
register int class;
{
    register LIST *temporary = find_list( _list_, identifier );

    if ( temporary == (LIST *)NULL )
        temporary = add_end_list( &_list_, identifier );

    if ( ( class & ( IMPLICIT ! EXPLICIT ) ) == EXPLICIT )
        temporary->type = type;
    else
        temporary->type = ZERO( temporary->type, implicit_type( identifier ) );

    temporary->structure = ZERO( temporary->structure, structure );
}

```

```

    temporary->list = ZERO( temporary->list, list );

    temporary->class |= class;
} /* add_list */

int length_list( list )
register LIST *list;
{
    register int length = 0;

    while ( list != (LIST *)NULL )
    {
        length++;

        list = list->next;
    }

    return( length );
} /* length_list */

void print_class( file, class )
register FILE *file;
register int class;
{
#ifdef DEBUG
    if ( ( class & ( IMPLICIT | EXPLICIT ) ) == EXPLICIT )
        fprintf( file, " EXPLICIT" );
    else
        fprintf( file, " IMPLICIT" );

    if ( ( class & ( LOCAL | GLOBAL ) ) == GLOBAL )
        fprintf( file, " GLOBAL" );
    else
        fprintf( file, " LOCAL" );

    if ( ( class & ( CONSTANT | VARIABLE ) ) == CONSTANT )
        fprintf( file, " CONSTANT" );
    else
        fprintf( file, " VARIABLE" );

    if ( ( class & ARRAY ) == ARRAY )
        fprintf( file, " ARRAY" );
    if ( ( class & FUNCTION ) == FUNCTION )
        fprintf( file, " FUNCTION" );
#else
    fprintf( file, " %x", class );
#endif
} /* print_class */

void print_list( file, list )
register FILE *file;
register LIST *list;
{
    while ( list != (LIST *)NULL )
    {
        fprintf( file, "%s %s %s", list->identifier, ZERO( list->type, "UNDEFINED" ),
ZERO( list->list, "{{0}{0}}" ) );

        print_class( file, list->class );

        fprintf( file, "\n" );

        list = list->next;
    }
} /* print_list */

FILE: usage/type/library/list.c

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```

extern char *parse( );
extern char *merge( );

char *list( input_list, delimiter )
register char *input_list;
register char *delimiter;
{
    register char *output_list;
    register char *list;
    register char *temporary;

    output_list = parse( input_list );
    list = parse( input_list );

    while ( list != (char *)0 )
    {
        temporary = merge( "%s%s", output_list, delimiter, list );
        free( output_list );
        free( list );

        output_list = temporary;
        list = parse( input_list );
    }

    return( output_list );
} /* list */

```

FILE: usage/type/library/main.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include <stdio.h>
#include <string.h>

extern FILE *yyin;
extern FILE *yyout;
extern char *default_integer;
extern char *default_logical;

#define PROGRAM argument[ 0 ]
#define INPUT_FILE argument[ 1 ]
#define OUTPUT_FILE argument[ 2 ]

int main( number_argument, argument )
int number_argument;
char *argument[ ];
{
loop:
    if ( strcmp( argument[ number_argument - 1 ], "-size=2" ) == 0 )
    {
        number_argument--;
        default_integer = "INTEGER*2";
        default_logical = "LOGICAL*2";
        goto loop;
    }

    if ( strcmp( argument[ number_argument - 1 ], "-size=4" ) == 0 )
    {
        number_argument--;
        default_integer = "INTEGER*4";
        default_logical = "LOGICAL*4";
        goto loop;
    }

    implicit_initialize( );

    if ( number_argument == 1 )
    {

```



```

yyin = stdin;
yyout = stdout;

yyvsparse( );
exit( 0 );
}

if ( number_argument == 3 )
{
    if ( ( yyin = fopen( INPUT_FILE, "r" ) ) == (FILE *)NULL )
    {
        fprintf( stderr, "%s: ERROR - unable to open input file '%s'\n", PROGRAM,
INPUT_FILE );
        exit( -1 );
    }

    if ( ( yyout = fopen( OUTPUT_FILE, "w" ) ) == (FILE *)NULL )
    {
        fprintf( stderr, "%s: ERROR - unable to open output file '%s'\n", PROGRAM,
OUTPUT_FILE );
        exit( -1 );
    }

   yyvsparse( );
    exit( 0 );
}

fprintf( stderr, "usage: %s <input file> <output file> [-size=2 or 4]\n", PROGRAM );
exit( 0 );
} /* main */

```

FILE: usage/type/library/merge.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include <stdio.h>
#include <string.h>
#include <malloc.h>

#define STRLEN( s ) ( strlen( s ) - 2 )

char *merge( string, a, b, c, d )
register char *string;
register char *a;
register char *b;
register char *c;
register char *d;
{
    register char *temporary = (char *)NULL;

    switch ( count( string, strlen( string ), '%' ) )
    {
        case 0:
            if ( ( temporary = (char *)malloc( strlen( string ) + 1 ) ) != (char *)NULL )
                sprintf( temporary, string );
            else
                fprintf( stderr, "ERROR: merge( %s )\n", string );
            break;

        case 1:
            if ( ( temporary = (char *)malloc( strlen( string ) + STRLEN( a ) + 1 ) ) !=
(char *)NULL )
                sprintf( temporary, string, a );
            else
                fprintf( stderr, "ERROR: merge( %s, %s )\n", string, a );
            break;

        case 2:
            if ( ( temporary = (char *)malloc( strlen( string ) + STRLEN( a ) + STRLEN( b
) + 1 ) ) != (char *)NULL )
                sprintf( temporary, string, a, b );

```

```

        else
            fprintf( stderr, "ERROR: merge( %s, %s, %s )\n", string, a, b );
            break;

        case 3:
            if ( ( temporary = (char *)malloc( strlen( string ) + STRLEN( a ) + STRLEN( b ) + STRLEN( c ) + 1 ) ) != (char *)NULL )
                sprintf( temporary, string, a, b, c );
            else
                fprintf( stderr, "ERROR: merge( %s, %s, %s, %s )\n", string, a, b, c );
                break;

        case 4:
            if ( ( temporary = (char *)malloc( strlen( string ) + STRLEN( a ) + STRLEN( b ) + STRLEN( c ) + STRLEN( d ) + 1 ) ) != (char *)NULL )
                sprintf( temporary, string, a, b, c, d );
            else
                fprintf( stderr, "ERROR: merge( %s, %s, %s, %s, %s )\n", string, a, b, c, d );
            );
            break;

        default:
            fprintf( stderr, "ERROR: merge( %s )\n", string );
            break;
    }

    return( temporary );
} /* merge */

```

FILE: usage/type/library/non\_blank.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```
#include <string.h>
```

```

char *non_blank( string )
register char *string;
{
    register int offset;
    register int length;

    length = strlen( string ) - 1;
    while ( ( string[ length ] == ' ' ) && ( string[ length ] != '\0' ) )
        string[ length-- ] = '\0';

    offset = 0;
    while ( ( string[ offset ] == ' ' ) && ( string[ offset ] != '\0' ) )
        string[ offset++ ] = '\0';

    strcpy( string, &string[ offset ] );

    if ( strlen( string ) != 0 )
        return( string );
    else
        return( 0 );
} /* non_blank */

```

FILE: usage/type/library/parse.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```
#include <string.h>
```

```
extern char *duplicate( );
```

```
char *parse( list )
register char *list;
{
    register int length = 0;
    register int brace = 0;
    register char *temporary = (char *)0;

    for (;;)
    {
        switch ( list[ length ] )
        {
            case '{':
                brace++;
                break;

            case '}':
                brace--;
                break;

            }

        if ( brace == 0 )
            break;

        length++;
    }

    if ( length != 0 )
    {
        list[ length ] = '\0';
        temporary = duplicate( list + 1 );
        strcpy( list, list + 1 + length );
    }
    else
    {
        if ( list[ length ] != '\0' )
        {
            temporary = duplicate( list );
            list[ length ] = '\0';
        }
    }

    return( temporary );
} /* parse */
```

```
FILE: usage/type/library/summary.c
```

```
/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */
```

```
#include <stdio.h>
#include "list.h"
```

```
extern FILE *yyin;
extern FILE *yyout;
```

```
void summary( )
{
    fprintf( yyout, "%d\n", length_list( _list_ ) );
    print_list( yyout, _list_ );
} /* summary */
```

```
FILE: usage/type/library/type.c
```

```
/*
 * Copyright 1991
 * Georgia Institute of Technology
```

```

* Computer Engineering Research Laboratory
* Author: Stephen R. Wachtel
*/

```

```

#include <string.h>

```

```

extern char *duplicate( );
extern char *merge( );

```

```

char *default_integer = "INTEGER*4";
char *default_logical = "LOGICAL*4";

```

```

char *type( type_name, type_length )
register char *type_name;
register char *type_length;
{
    if ( type_length != (char *)0 )
        return( merge( "%s%s", type_name, type_length ) );

    if ( strcmp( type_name, "CHARACTER" ) == 0 )
        return( duplicate( "CHARACTER*1" ) );

    if ( strcmp( type_name, "COMPLEX" ) == 0 )
        return( duplicate( "COMPLEX*8" ) );

    if ( strcmp( type_name, "DOUBLE_PRECISION" ) == 0 )
        return( duplicate( "REAL*8" ) );

    if ( strcmp( type_name, "INTEGER" ) == 0 )
        return( duplicate( default_integer ) );

    if ( strcmp( type_name, "LOGICAL" ) == 0 )
        return( duplicate( default_logical ) );

    if ( strcmp( type_name, "REAL" ) == 0 )
        return( duplicate( "REAL*4" ) );

    return( duplicate( type_name ) );
} /* type */

```

```

FILE: usage/type/library/uppercase.c

```

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```

char *uppercase( string )
register char *string;
{
    register int index = 0;

    while ( string[ index ] != '\0' )
    {
        string[ index ] = toupper( string[ index ] );
        index++;
    }

    return( string );
} /* uppercase */

```

```

FILE: usage/type/library/yyerror.c

```

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```
#include <stdio.h>

extern int yylineno;

void yyerror( string )
register char *string;
{
    fprintf( stderr, "line %d, %s\n", yylineno, string );

    exit( -1 );
} /* yyerror */
```

FILE: usage/type/library/yygetc.c

```
/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include <stdio.h>
#include <ctype.h>

extern int yylineno;

int tab( length )
register int length;
{
    while ( length-- != 0 )
        yyunput( ' ' );

    return( ' ' );
} /* tab */

int yygetc( file )
register FILE *file;
{
    int c;
    int column[ 6 ];

loop:
    if ( ( c = getc( file ) ) == '\t' )
        c = tab( 6 );

    if ( c != '\n' )
        return( c );

    if ( ( column[ 0 ] = getc( file ) ) != ' ' )
        goto abort_0;
    if ( ( column[ 1 ] = getc( file ) ) != ' ' )
        goto abort_1;
    if ( ( column[ 2 ] = getc( file ) ) != ' ' )
        goto abort_2;
    if ( ( column[ 3 ] = getc( file ) ) != ' ' )
        goto abort_3;
    if ( ( column[ 4 ] = getc( file ) ) != ' ' )
        goto abort_4;
    if ( isspace( column[ 5 ] = getc( file ) ) )
        goto abort_5;

    yylineno++;
    goto loop;

abort_5:
    if ( column[ 5 ] == '\t' )
        tab( 1 );
    else
    {
        yyunput( column[ 5 ] );
        if ( column[ 5 ] == '\n' )
            yylineno++;
    }
```

```

    }
abort_4:
    if ( column[ 4 ] == '\t' )
        tab( 2 );
    else
    {
        yyunput( column[ 4 ] );
        if ( column[ 4 ] == '\n' )
            yylineno++;
    }
abort_3:
    if ( column[ 3 ] == '\t' )
        tab( 3 );
    else
    {
        yyunput( column[ 3 ] );
        if ( column[ 3 ] == '\n' )
            yylineno++;
    }
abort_2:
    if ( column[ 2 ] == '\t' )
        tab( 4 );
    else
    {
        yyunput( column[ 2 ] );
        if ( column[ 2 ] == '\n' )
            yylineno++;
    }
abort_1:
    if ( column[ 1 ] == '\t' )
        tab( 5 );
    else
    {
        yyunput( column[ 1 ] );
        if ( column[ 1 ] == '\n' )
            yylineno++;
    }
abort_0:
    if ( column[ 0 ] == '\t' )
        tab( 6 );
    else
    {
        yyunput( column[ 0 ] );
        if ( column[ 0 ] == '\n' )
            yylineno++;
    }
    return( 0 );
} /* yygetc */

```

FILE: usage/type/library/yywrap.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```

int yywrap( )
{
    return( 1 );
} /* yywrap */

```

FILE: usage/type/scanner.l

```

%{
/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 */

```

```

* Author: Stephen R. Wachtei
*/
%)

```

```

%a 10000
%e 10000
%k 10000
%n 10000
%o 10000
%p 10000

```

```

a [aA]
b [bB]
c [cC]
d [dD]
e [eE]
f [fF]
g [gG]
h [hH]
i [iI]
j [jJ]
k [kK]
l [lL]
m [mM]
n [nN]
o [oO]
p [pP]
q [qQ]
r [rR]
s [sS]
t [tT]
u [uU]
v [vV]
w [wW]
x [xX]
y [yY]
z [zZ]

```

```

%{
#include "grammar.h"
extern char *yyival;

```

```

#undef YYLMAX
#define YYLMAX (256*20)

```

```

extern char *duplicate( );
extern char *hollerith( );
extern char *non_blank( );
extern char *uppercase( );
%}

```

```

%%

```

```

^[\\*cC].*[\\n]
^[\\_ '\\n] {
#ifdef DEBUG
    ECHO;
#endif
    yyival = duplicate( yytext );
    return( COMMENT );
}

```

```

[\\ ] {
#ifdef DEBUG
    ECHO;
#endif
    /* return( '\\ ' ) */;
}

```

```

[\\&] {
#ifdef DEBUG
    ECHO;

```

```

#endif
    return( '\&' );
}

[\] {
#ifdef DEBUG
    ECHO;
#endif
    return( '\(' );
}

[\)] {
#ifdef DEBUG
    ECHO;
#endif
    return( '\)' );
}

[\*] {
#ifdef DEBUG
    ECHO;
#endif
    return( '\*' );
}

[\*][\*] {
#ifdef DEBUG
    ECHO;
#endif
    return( EXPONENTIATE );
}

[\+] {
#ifdef DEBUG
    ECHO;
#endif
    return( '\+' );
}

[\,] {
#ifdef DEBUG
    ECHO;
#endif
    return( '\,' );
}

[\-] {
#ifdef DEBUG
    ECHO;
#endif
    return( '\-' );
}

[\.] {
#ifdef DEBUG
    ECHO;
#endif
    return( '\.' );
}

[/] {
#ifdef DEBUG
    ECHO;
#endif
    return( '/' );
}

[:] {
#ifdef DEBUG
    ECHO;
#endif

```



```

    return( '\t' );
}

[\=] {
#ifdef DEBUG
    ECHO;
#endif
    return( '\=' );
}

[\n] {
#ifdef DEBUG
    ECHO;
#endif
    /* return( '\n' ) */;
}

[\t] {
#ifdef DEBUG
    ECHO;
#endif
    /* return( '\t' ) */;
}

[\.]{a}{n}{d}[\.] {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_AND );
}

[\.]{e}{q}[\.] {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_EQ );
}

[\.]{e}{q}{v}[\.] {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_EQV );
}

[\.]{f}{a}{l}{s}{e}[\.] {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_FALSE );
}

[\.]{g}{e}[\.] {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_GE );
}

[\.]{g}{t}[\.] {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_GT );
}

[\.]{l}{e}[\.] {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_LE );
}

```

```

}

[\\.] {l} {t} [\\.] {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_LT );
}

[\\.] {n} {e} [\\.] {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_NE );
}

[\\.] {n} {e} {q} {v} [\\.] {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_NEQV );
}

[\\.] {n} {o} {t} [\\.] {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_NOT );
}

[\\.] {o} {r} [\\.] {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_OR );
}

[\\.] {t} {r} {u} {e} [\\.] {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_TRUE );
}

{a} {s} {s} {i} {g} {n} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_ASSIGN );
}

{b} {a} {c} {k} {s} {p} {a} {c} {e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_BACKSPACE );
}

{b} {l} {o} {c} {k} [\\ ] * {d} {a} {t} {a} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_BLOCK_DATA );
}

{c} {a} {l} {l} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_CALL );
}

```

```

(c){h}{a}{r}{a}{c}{t}{e}{r} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_CHARACTER );
}

(c){l}{o}{s}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_CLOSE );
}

(c){o}{m}{m}{o}{n} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_COMMON );
}

(c){o}{m}{p}{l}{e}{x} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_COMPLEX );
}

(c){o}{n}{t}{i}{n}{u}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_CONTINUE );
}

(d){a}{t}{a} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_DATA );
}

(d){i}{m}{e}{n}{s}{i}{o}{n} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_DIMENSION );
}

(d){o} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_DO );
}

(d){o}{u}{b}{l}{e}{[ \ ]*{p}{r}{e}{c}{i}{s}{i}{o}{n} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_DOUBLE_PRECISION );
}

(e){l}{s}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_ELSE );
}

```

```

{e}{l}{s}{e}{\ }*{i}{f} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_ELSE_IF );
}

{e}{n}{d} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_END );
}

{e}{n}{d}{\ }*{i}{f} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_END_IF );
}

{e}{n}{d}{f}{i}{l}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_ENDFILE );
}

{e}{n}{t}{r}{y} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_ENTRY );
}

{e}{q}{u}{i}{v}{a}{l}{e}{n}{c}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_EQUIVALENCE );
}

{e}{x}{t}{e}{r}{n}{a}{l} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_EXTERNAL );
}

{f}{o}{r}{m}{a}{t}{.*} {
#ifdef DEBUG
    ECHO;
#endif
    yy1val = duplicate( yytext );
    return( RW_FORMAT );
}

{f}{u}{n}{c}{t}{i}{o}{n} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_FUNCTION );
}

{g}{o}{\ }*{t}{o} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_GO_TO );
}

```

```

{i}{f} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_IF );
}

{i}{m}{p}{l}{i}{c}{i}{t} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_IMPLICIT );
}

{i}{n}{c}{l}{u}{d}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_INCLUDE );
}

{i}{n}{q}{u}{i}{r}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_INQUIRE );
}

{i}{n}{t}{e}{g}{e}{r} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_INTEGER );
}

{i}{n}{t}{r}{i}{n}{s}{i}{c} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_INTRINSIC );
}

{l}{o}{g}{i}{c}{a}{l} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_LOGICAL );
}

{n}{a}{m}{e}{l}{i}{s}{t} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_NAMELIST );
}

{o}{p}{e}{n} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_OPEN );
}

{p}{a}{r}{a}{m}{e}{t}{e}{r} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_PARAMETER );
}

```

```

{p}{a}{u}{s}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_PAUSE );
}

{p}{r}{i}{n}{t} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_PRINT );
}

{p}{r}{o}{g}{r}{a}{m} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_PROGRAM );
}

{r}{e}{a}{d} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_READ );
}

{r}{e}{a}{l} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_REAL );
}

{r}{e}{t}{u}{r}{n} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_RETURN );
}

{r}{e}{w}{i}{n}{d} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_REWIND );
}

{s}{a}{v}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_SAVE );
}

{s}{t}{o}{p} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_STOP );
}

{s}{u}{b}{r}{o}{u}{t}{i}{n}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_SUBROUTINE );
}

{t}{h}{e}{n} {

```

```

#ifdef DEBUG
    ECHO;
#endif
    return( RW_THEN );
}

{t}{o} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_TO );
}

{w}{r}{i}{t}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_WRITE );
}

{u}{n}{d}{e}{f}{i}{n}{e}{d} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_UNDEFINED );
}

[%a-zA-Z][_a-zA-Z0-9]* {
#ifdef DEBUG
    ECHO;
#endif
    yy1val = duplicate( uppercase( yytext ) );
    return( IDENTIFIER );
}

^[0-9][0-9][0-9][0-9][0-9][\ ] {
#ifdef DEBUG
    ECHO;
#endif
    yy1val = duplicate( non_blank( yytext ) );
    return( LABEL );
}

[0-9]+ |
[0-9]+/\.[a-zA-Z]+\.\. {
#ifdef DEBUG
    ECHO;
#endif
    yy1val = duplicate( yytext );
    return( INTEGER );
}

[0-9]+\.[0-9]*([eE][\+\-]?[0-9]+)? |
[0-9]*\.[0-9]+([eE][\+\-]?[0-9]+)? |
[0-9]+([eE][\+\-]?[0-9]+)? {
#ifdef DEBUG
    ECHO;
#endif
    yy1val = duplicate( yytext );
    return( REAL );
}

[0-9]+\.[0-9]*([dD][\+\-]?[0-9]+)? |
[0-9]*\.[0-9]+([dD][\+\-]?[0-9]+)? |
[0-9]+([dD][\+\-]?[0-9]+)? {
#ifdef DEBUG
    ECHO;
#endif
    yy1val = duplicate( yytext );
    return( DOUBLE_PRECISION );
}

```

```

\'[^\']*\' |
\"[^\"]*\" {
#ifdef DEBUG
    ECHO;
#endif
    yytext[ 0 ] = '\\';
    yytext[ strlen( yytext ) - 1 ] = '\\';
    yylval = duplicate( yytext );
    return( STRING );
}

[0-9]+[hH] {
#ifdef DEBUG
    ECHO;
#endif
    yylval = duplicate( hollerith( yytext, '\\\"' ) );
    return( HOLLERITH );
}

```

FILE: usage/type/statement/Makefile

```

#
# Copyright 1991
# Georgia Institute of Technology
# Computer Engineering Research Laboratory
# Author: Stephen R. Wachtel
#

```

```

CC = cc -g
INCLUDE = ../include
CFLAGS = -I$(INCLUDE)
LIBRARY = statement.a

```

```

OBJECTS = \
    common_statement.o \
    declaration_statement.o \
    dimension_statement.o \
    do_statement.o \
    end_statement.o \
    function_statement.o \
    implicit_statement.o \
    parameter_statement.o

```

```

$(LIBRARY): $(OBJECTS)
    rm -f $(LIBRARY)
    ar crv $(LIBRARY) $(OBJECTS)
    ranlib $(LIBRARY)

```

```

.SUFFIXES: .c .o
.c.o:
    $(CC) -c $(CFLAGS) $<

```

```

clean:
    rm -f $(LIBRARY) $(OBJECTS)

```

FILE: usage/type/statement/common\_statement.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```

#include "list.h"
#include "class.h"

```

```

extern char *duplicate( );
extern char *parse( );

```



```

void common_statement( common_name, common_list )
register char *common_name;
register char *common_list;
{
    register char *common;
    register char *identifier;
    register char *subscript_list;

    if ( common_name == (char *)0 )
        common_name = duplicate( "_COMMON_" );

    add_list( 0, common_name, 0, 0, IMPLICIT | GLOBAL | VARIABLE );

    while ( common = parse( common_list ) )
    {
        identifier = parse( common );
        subscript_list = parse( common );

        if ( subscript_list == (char *)0 )
            add_list( common_name, identifier, 0, subscript_list, IMPLICIT | LOCAL |
VARIABLE );
        else
            add_list( common_name, identifier, 0, subscript_list, IMPLICIT | LOCAL |
VARIABLE | ARRAY );
    }
} /* common_statement */

```

FILE: usage/type/statement/declaration\_statement.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen P. Wachtel
 */

```

```

#include "list.h"
#include "class.h"

```

```

extern char *parse();

```

```

void declaration_statement( type, declaration_list )
register char *type;
register char *declaration_list;
{
    register char *declaration;
    register char *identifier;
    register char *subscript_list;

    while ( declaration = parse( declaration_list ) )
    {
        identifier = parse( declaration );
        subscript_list = parse( declaration );

        if ( subscript_list == (char *)0 )
            add_list( 0, identifier, type, subscript_list, EXPLICIT | LOCAL | VARIABLE );
        else
            add_list( 0, identifier, type, subscript_list, EXPLICIT | LOCAL | VARIABLE |
ARRAY );
    }
} /* declaration_statement */

```

FILE: usage/type/statement/dimension\_statement.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```

#include "list.h"
#include "class.h"

extern char *parse( );

void dimension_statement( dimension_list )
register char *dimension_list;
{
    register char *dimension;
    register char *identifier;
    register char *subscript_list;

    while ( dimension = parse( dimension_list ) )
    {
        identifier = parse( dimension );
        subscript_list = parse( dimension );

        add_list( 0, identifier, 0, subscript_list, IMPLICIT | LOCAL | VARIABLE | ARRAY );
    }
} /* dimension_statement */

```

FILE: usage/type/statement/do\_statement.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```

#include "list.h"
#include "class.h"

```

```

void do_statement( label, identifier, expression_list )
register char *label;
register char *identifier;
register char *expression_list;
{
    add_list( 0, identifier, 0, 0, IMPLICIT | LOCAL | VARIABLE );
} /* do_statement */

```

FILE: usage/type/statement/end\_statement.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```

void end_statement( )
{
    implicit_initialize( );
} /* end_statement */

```

FILE: usage/type/statement/function\_statement.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```

#include "list.h"
#include "class.h"

```

```

void function_statement( type, identifier, formal_argument_list )

```

```

register char *type;
register char *identifier;
register char *formal_argument_list;
{
    if ( type == (char *)0 )
        add_list( 0, identifier, type, 0, IMPLICIT | GLOBAL | VARIABLE | FUNCTION );
    else
        add_list( 0, identifier, type, 0, EXPLICIT | GLOBAL | VARIABLE | FUNCTION );
} /* function_statement */

```

FILE: usage/type/statement/implicit\_statement.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```
extern char *parse( );
```

```

void implicit_statement( type, implicit_list )
register char *type;
register char *implicit_list;
{
    register char *implicit;
    register char *lower_bound;
    register char *upper_bound;

    while ( implicit = parse( implicit_list ) )
    {
        lower_bound = parse( implicit );
        upper_bound = parse( implicit );

        type_implicit( type, lower_bound, upper_bound );
    }
} /* implicit_statement */

```

FILE: usage/type/statement/parameter\_statement.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```

#include "list.h"
#include "class.h"

```

```
extern char *parse( );
```

```

void parameter_statement( parameter_list )
register char *parameter_list;
{
    register char *parameter;
    register char *identifier;
    register char *expression;

    while ( parameter = parse( parameter_list ) )
    {
        identifier = parse( parameter );
        expression = parse( parameter );

        add_list( 0, identifier, 0, 0, IMPLICIT | LOCAL | CONSTANT );
    }
} /* parameter_statement */

```

FILE: usage/usage/Makefile

```

#
# Copyright 1991
# Georgia Institute of Technology
# Computer Engineering Research Laboratory
# Author: Stephen R. Wachtel
#

default:  usage

CC = cc -g
INCLUDE = include
CFLAGS = -I$(INCLUDE)
LIBRARY = library/library.a

OBJECTS = \
    $(INCLUDE)/grammar.h \
    *grammar.[co] \
    *scanner.[co] \
    yytrace.[co] \
    y.output

PROGRAMS = \
    *usage

grammar.c: grammar.y
    yacc -dv grammar.y
    mv y.tab.h $(INCLUDE)/grammar.h
    mv y.tab.c grammar.c

scanner.c: scanner.l
    lex -vt scanner.l | sed 's/getc/yygetc/' >scanner.c

scanner.o: scanner.c $(INCLUDE)/grammar.h
    $(CC) $(CFLAGS) -c scanner.c

grammar.o: grammar.c
    $(CC) $(CFLAGS) -c grammar.c

usage: grammar.o scanner.o $(LIBRARY)
    $(CC) -o usage grammar.o scanner.o $(LIBRARY)

sgrammar.c: grammar.c yytoken.awk
    awk -f yytoken.awk <grammar.c >sgrammar.c

sgrammar.o: sgrammar.c
    $(CC) $(CFLAGS) -c sgrammar.c

susage: sgrammar.o scanner.o $(LIBRARY)
    $(CC) -o susage sgrammar.o scanner.o $(LIBRARY)

dscanner.c: scanner.c
    cp scanner.c dscanner.c

dscanner.o: dscanner.c $(INCLUDE)/grammar.h
    $(CC) $(CFLAGS) -DDEBUG -c dscanner.c

dusage: grammar.o dscanner.o $(LIBRARY)
    $(CC) -o dusage grammar.o dscanner.o $(LIBRARY)

tgrammar.c: grammar.c
    sed 's/yystack:/& yytrace(yystate);/' <grammar.c >tgrammar.c

tgrammar.o: tgrammar.c
    $(CC) $(CFLAGS) -c tgrammar.c

*usage: tgrammar.o scanner.o yytrace.o $(LIBRARY)
    $(CC) -o tusage tgrammar.o scanner.o yytrace.o $(LIBRARY)

yytrace.c: grammar.c yytrace.awk
    awk -f yytrace.awk <y.output >yytrace.c

```

```
yytrace.o: yytrace.c
$(CC) $(CFLAGS) -c yytrace.c
```

```
clean:
    cd library; make clean
    rm -f $(PROGRAMS) $(OBJECTS)
```

```
FILE: usage/usage/grammar.y
```

```
/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */
```

```
/*
 * FORTRAN 77
 */
```

```
%token RW_AND
%token RW_ASSIGN
%token RW_BACKSPACE
%token RW_BLOCK_DATA
%token RW_CALL
%token RW_CHARACTER
%token RW_CLOSE
%token RW_COMMON
%token RW_COMPLEX
%token RW_CONTINUE
%token RW_DATA
%token RW_DIMENSION
%token RW_DO
%token RW_DO_WHILE
%token RW_DOUBLE_PRECISION
%token RW_ELSE
%token RW_ELSE_IF
%token RW_END
%token RW_END_DO
%token RW_END_IF
%token RW_ENDFILE
%token RW_ENTRY
%token RW_EQ
%token RW_EQUIVALENCE
%token RW_EQV
%token RW_EXTERNAL
%token RW_FALSE
%token RW_FORMAT
%token RW_FUNCTION
%token RW_GE
%token RW_GO_TO
%token RW_GT
%token RW_IF
%token RW_IMPLICIT
%token RW_INCLUDE
%token RW_INQUIRE
%token RW_INTEGER
%token RW_INTRINSIC
%token RW_LE
%token RW_LOGICAL
%token RW_LT
%token RW_NAMELIST
%token RW_NE
%token RW_NEQV
%token RW_NOT
%token RW_OPEN
%token RW_OR
%token RW_PARAMETER
%token RW_PAUSE
%token RW_PRINT
%token RW_PROGRAM
%token RW_READ
%token RW_REAL
%token RW_RETURN
%token RW_REWIND
%token RW_SAVE
```

```

%token RW_STOP
%token RW_SUBROUTINE
%token RW_THEN
%token RW_TO
%token RW_TRUE
%token RW_WRITE
%token RW_UNDEFINED

%token CONCATENATE
%token COMMENT
%token DOUBLE_PRECISION
%token EXPONENTIATE
%token HOLLERITH
%token IDENTIFIER
%token INTEGER
%token LABEL
%token REAL
%token STRING

%left ','
%nonassoc ':'
%right '='
%left RW_EQV RW_NEQV
%left RW_OR
%left RW_AND
%left RW_NOT
%nonassoc RW_EQ RW_NE RW_LT RW_LE RW_GT RW_GE
%left CONCATENATE
%left '+' '-'
%left '*' '/'
%right EXPONENTIATE
%left SIGN

```

```

%{
typedef char *POINTER;
#define YYSTYPE POINTER

#include "usage.h"
static int usage = REF;
static int enable = 0;
%}

```

```

%%

```

```

program:
    optional_statement_list
    {
        summary( );
    }
    ;

optional_statement_list:
    /* NULL */
    ;
    statement_list
    ;

statement_list:
    statement
    ;
    statement_list statement
    ;

statement:
    comment_statement
    ;
    label_unlabeled_statement
    ;

comment_statement:
    COMMENT
    ;

```

```

        if ( strcmp( $1, "**LOOP* PROLOGUE\n" ) == 0 )
            enable = 1;
        if ( strcmp( $1, "**LOOP* EPILOGUE\n" ) == 0 )
            enable = 0;
    }

label:
    LABEL
;

unlabeled_statement:
    include_statement
    |
    program_statement
    |
    block_data_statement
    |
    function_statement
    |
    subroutine_statement
    |
    entry_statement
    |
    end_statement
    |
    specification_statement
    |
    executable_statement
    |
    format_statement
;

include_statement:
    RW_INCLUDE character_constant
;

program_statement:
    RW_PROGRAM program_identifier
;

program_identifier:
    IDENTIFIER
    {
        begin_block( $1 );
        enable = 0;
    }
;

block_data_statement:
    RW_BLOCK_DATA block_data_identifier
;

block_data_identifier:
    IDENTIFIER
    {
        begin_block( $1 );
        enable = 1;
    }
;

function_statement:
    RW_FUNCTION function_identifier optional_formal_argument_list
    |
    type RW_FUNCTION function_identifier optional_formal_argument_list
;

function_identifier:
    IDENTIFIER
    {
        begin_block( $1 );
        enable = 1;
    }

```

```

    ;
}

subroutine_statement:
    RW_SUBROUTINE subroutine_identifier
    |
    RW_SUBROUTINE subroutine_identifier optional_formal_argument_list
    ;

subroutine_identifier:
    IDENTIFIER
    {
        begin_block( $1 );
        enable = 1;
    }
    ;

entry_statement:
    RW_ENTRY entry_identifier
    |
    RW_ENTRY entry_identifier optional_formal_argument_list
    ;

entry_identifier:
    IDENTIFIER
    ;

optional_formal_argument_list:
    '(' ',' ')'
    |
    '(' formal_argument_list ')'
    ;

formal_argument_list:
    formal_argument
    |
    formal_argument_list ',' formal_argument
    ;

formal_argument:
    IDENTIFIER
    {
        if ( enable )
            add_formal_argument_list( $1 );
    }
    |
    formal_argument_alterate_return
    {
        if ( enable )
            add_formal_argument_list( $1 );
    }
    ;

formal_argument_alterate_return:
    '*'
    {
        $$ = "";
    }
    ;

end_statement:
    RW_END
    {
        end_block( );
        enable = 0;
    }
    ;

specification_statement:
    external_statement
    |

```



```

intrinsic_statement
|
parameter_statement
|
dimension_statement
|
declaration_statement
|
save_statement
|
common_statement
|
equivalence_statement
|
implicit_statement
|
data_statement
|
namelist_statement
;

external_statement:
    RW_EXTERNAL external_list
;

external_list:
    external
|
    external_list ',' external
;

external:
    IDENTIFIER
;

intrinsic_statement:
    RW_INTRINSIC intrinsic_list
;

intrinsic_list:
    intrinsic
|
    intrinsic_list ',' intrinsic
;

intrinsic:
    IDENTIFIER
;

parameter_statement:
    RW_PARAMETER '(' parameter_list ')'
;

parameter_list:
    parameter
|
    parameter_list ',' parameter
;

parameter:
    parameter_identifier '=' expression
;

parameter_identifier:
    IDENTIFIER
    {
        if ( enable )
            usage_identifier( $1, SET );
    }
;

```

```

dimension_statement:
    RW_DIMENSION dimension_list
    ;

dimension_list:
    dimension
    |
    dimension_list ',' dimension
    ;

dimension:
    IDENTIFIER '(' subscript_list ')'
    ;

subscript_list:
    subscript
    |
    subscript_list ',' subscript
    ;

subscript:
    upper_bound
    |
    lower_bound ':' upper_bound
    ;

lower_bound:
    INTEGER
    |
    INTEGER '*' IDENTIFIER
    |
    '+' INTEGER
    |
    '-' INTEGER
    |
    IDENTIFIER
    |
    IDENTIFIER '*' INTEGER
    |
    '+' IDENTIFIER
    |
    '-' IDENTIFIER
    ;

upper_bound:
    lower_bound
    |
    upper_bound_adjustable
    ;

upper_bound_adjustable:
    '*'
    ;

declaration_statement:
    type declaration_list
    ;

declaration_list:
    declaration
    |
    declaration_list ',' declaration
    ;

declaration:
    IDENTIFIER optional_type_length
    |
    IDENTIFIER '(' subscript_list ')' optional_type_length
    ;

```

```

type:
    type_name optional_type_length
    ;

```

```

type_name:
    RW_CHARACTER
    |
    RW_COMPLEX
    |
    RW_DOUBLE_PRECISION
    |
    RW_INTEGER
    |
    RW_LOGICAL
    |
    RW_REAL
    |
    RW_UNDEFINED
    ;

```

```

optional_type_length:
    /* NULL */
    |
    type_length
    ;

```

```

type_length:
    '*' INTEGER
    |
    '*' type_length_adjustable
    ;

```

```

type_length_adjustable:
    '(' '*' ')'
    ;

```

```

save_statement:
    RW_SAVE optional_save_list
    ;

```

```

optional_save_list:
    /* NULL */
    |
    save_list
    ;

```

```

save_list:
    save
    |
    save_list ',' save
    ;

```

```

save:
    IDENTIFIER
    |
    common_name
    ;

```

```

common_statement:
    RW_COMMON optional_common_name common_variable_list
    ;

```

```

optional_common_name:
    /* NULL */
    |
    common_name
    ;

```

```

common_name:

```

```

        '/' optional_identifier '/'
    ;

optional_identifier:
    /* NULL */
    |
    IDENTIFIER
    ;

common_variable_list:
    common_variable
    |
    common_variable_list ',' common_variable
    ;

common_variable:
    IDENTIFIER
    |
    IDENTIFIER '(' subscript_list ')'
    ;

equivalence_statement:
    RW_EQUIVALENCE equivalence_list
    ;

equivalence_list:
    equivalence
    |
    equivalence_list ',' equivalence
    ;

equivalence:
    '(' equivalence_variable_list ')'
    ;

equivalence_variable_list:
    equivalence_variable
    |
    equivalence_variable_list ',' equivalence_variable
    ;

equivalence_variable:
    IDENTIFIER
    |
    IDENTIFIER '(' subscript_list ')'
    ;

implicit_statement:
    RW_IMPLICIT type '(' implicit_list ')'
    ;

implicit_list:
    implicit
    |
    implicit_list ',' implicit
    ;

implicit:
    IDENTIFIER
    |
    IDENTIFIER '-' IDENTIFIER
    ;

namelist_statement:
    RW_NAMELIST namelist_name namelist_list
    ;

namelist_name:

```

```

        '/' IDENTIFIER '/'
    ;

namelist_list:
    namelist
    |
    namelist_list ',' namelist
    ;

namelist:
    IDENTIFIER
    ;

data_statement:
    RW_DATA data_list
    ;

data_list:
    data
    |
    data_list optional_comma data
    ;

data:
    data_variable_list '/' data_constant_list '/'
    ;

data_variable_list:
    data_variable
    |
    data_variable_list ',' data_variable
    ;

data_variable:
    variable
    {
        if ( enable )
            usage_identifier( $1, SET );
    }
    |
    data_implied_do_list
    ;

data_implied_do_list:
    '(' data_variable_list ',' IDENTIFIER '=' expression_list ')'
    ;

data_constant_list:
    data_constant
    |
    data_constant_list ',' data_constant
    ;

data_constant:
    data_value
    |
    IDENTIFIER '*' data_value
    |
    INTEGER '*' data_value
    ;

data_value:
    IDENTIFIER
    |
    character_constant
    |
    logical_constant
    |
    numerical_constant
    ;

```

```

expression:
    simple_expression
    {
        $$ = $1;
    }
    |
    parenthesis_expression
    {
        $$ = $1;
    }
    ;

simple_expression:
    variable
    {
        $$ = $1;
    }
    |
    constant
    {
        $$ = $1;
    }
    |
    unary_expression
    {
        $$ = $1;
    }
    |
    arithmetic_expression
    {
        $$ = $1;
    }
    |
    character_expression
    {
        $$ = $1;
    }
    |
    relational_expression
    {
        $$ = $1;
    }
    |
    logical_expression
    {
        $$ = $1;
    }
    ;

parenthesis_expression:
    '(' expression ')'
    {
        $$ = "";
    }
    ;

arithmetic_expression:
    expression '+' expression %prec '+'
    {
        $$ = "";
    }
    |
    expression '-' expression %prec '-'
    {
        $$ = "";
    }
    |
    expression '*' expression %prec '*'
    {
        $$ = "";
    }
    |
    expression '/' expression %prec '/'
    {
        $$ = "";
    }
    ;

```

```

|
|   expression EXPONENTIATE expression %prec EXPONENTIATE
|   {
|       $$ = "";
|   }
|
;
```

```

character_expression:
|   expression '/' '/' expression %prec CONCATENATE
|   {
|       $$ = "";
|   }
|
;
```

```

relational_expression:
|   expression RW_EQ expression %prec RW_EQ
|   {
|       $$ = "";
|   }
|
|   expression RW_NE expression %prec RW_NE
|   {
|       $$ = "";
|   }
|
|   expression RW_LT expression %prec RW_LT
|   {
|       $$ = "";
|   }
|
|   expression RW_LE expression %prec RW_LE
|   {
|       $$ = "";
|   }
|
|   expression RW_GT expression %prec RW_GT
|   {
|       $$ = "";
|   }
|
|   expression RW_GE expression %prec RW_GE
|   {
|       $$ = "";
|   }
|
;
```

```

logical_expression:
|   expression RW_AND expression %prec RW_AND
|   {
|       $$ = "";
|   }
|
|   expression RW_OR expression %prec RW_OR
|   {
|       $$ = "";
|   }
|
|   expression RW_EQV expression %prec RW_EQV
|   {
|       $$ = "";
|   }
|
|   expression RW_NEQV expression %prec RW_NEQV
|   {
|       $$ = "";
|   }
|
;
```

```

unar_expression:
|   '+' expression %prec SIGN
|   {
|       $$ = "";
|   }
|
|   '-' expression %prec SIGN
|   {
|       $$ = "";
|   }
|
;
```

```

    }
    |
    RW_NOT expression %prec RW_NOT
    {
        $$ = "";
    }
    ;

variable:
    identifier
    {
        $$ = $1;
    }
    |
    identifier string_subset
    {
        $$ = $1;
    }
    |
    array
    {
        $$ = $1;
    }
    ;

array:
    identifier '(' optional_expression_list ')'
    {
        $$ = $1;
    }
    |
    identifier '(' optional_expression_list ')' string_subset
    {
        $$ = $1;
    }
    ;

identifier:
    IDENTIFIER
    {
        if ( enable )
            usage_identifier( $1, usage );
        if ( usage == SET ) usage = REF;
        $$ = $1;
    }
    ;

optional_expression_list:
    /* NULL */
    |
    expression_list
    ;

expression_list:
    expression
    |
    expression_list ',' expression
    ;

string_subset:
    '(' optional_expression ':' optional_expression ')'
    ;

optional_expression:
    /* NULL */
    |
    expression
    ;

constant:
    logical_constant
    {

```



```

        $$ = $1;
    }
    |
    character_constant
    {
        $$ = $1;
    }
    |
    unsigned_numerical_constant
    {
        $$ = $1;
    }
    ;

logical_constant:
    RW_TRUE
    {
        $$ = "";
    }
    |
    RW_FALSE
    {
        $$ = "";
    }
    ;

character_constant:
    HOLLERITH
    {
        $$ = "";
    }
    |
    STRING
    {
        $$ = "";
    }
    ;

unsigned_numerical_constant:
    INTEGER
    {
        $$ = "";
    }
    |
    REAL
    {
        $$ = "";
    }
    |
    DOUBLE_PRECISION
    {
        $$ = "";
    }
    ;

numerical_constant:
    unsigned_numerical_constant
    |
    '+' unsigned_numerical_constant %prec SIGN
    |
    '-' unsigned_numerical_constant %prec SIGN
    ;

executable_statement:
    do_statement
    |
    do_while_statement
    |
    end_do_statement
    |
    logical_if_statement
    |
    block_if_statement
    |
    else_statement
    ;

```

```

        else_if_statement
    |
    end_if_statement
    |
    subset_executable_statement
    ;

do_statement:
    RW_DO optional_integer do_identifier '=' expression_list
    ;

do_identifier:
    IDENTIFIER
    {
        if ( enable )
            usage_identifier( $1, SET );
    }
    ;

optional_integer:
    /* NULL */
    |
    INTEGER
    ;

do_while_statement:
    RW_DO_WHILE '(' expression ')'
    ;

end_do_statement:
    RW_END_DO
    ;

logical_if_statement:
    if_expression subset_executable_statement
    ;

if_expression:
    RW_IF '(' expression ')'
    ;

block_if_statement:
    RW_IF '(' expression ')' RW_THEN
    ;

else_statement:
    RW_ELSE
    ;

else_if_statement:
    RW_ELSE_IF '(' expression ')' RW_THEN
    ;

end_if_statement:
    RW_END_IF
    ;

subset_executable_statement:
    assignment_statement
    |
    assign_statement
    |
    arithmetic_if_statement
    |
    continue_statement
    |
    call_statement
    |
    return_statement

```

```

| unconditional_go_to_statement
| computed_go_to_statement
| assigned_go_to_statement
| stop_statement
| pause_statement
| io_statement
;

assignment_statement:
    { if ( usage == REF ) usage = SET; ; variable '=' expression
;

assign_statement:
    RW_ASSIGN INTEGER RW_TO IDENTIFIER
;

arithmetic_if_statement:
    RW_IF '(' expression ')' integer_list
;

continue_statement:
    RW_CONTINUE
;

call_statement:
    RW_CALL call_identifier
|
    RW_CALL call_identifier optional_actual_argument_list
;

call_identifier:
    IDENTIFIER
    {
        if ( enable )
            add_call_list( $1 );
    }
;

optional_actual_argument_list:
    '(' ',' ')'
|
    '(' { usage &= ~REF; } actual_argument_list { usage != REF; } ')'
;

actual_argument_list:
    actual_argument
|
    actual_argument_list ',' actual_argument
;

actual_argument:
    expression
    {
        if ( enable )
            add_actual_argument_list( $1 );
    }
|
    actual_argument_alternate_return
    {
        if ( enable )
            add_actual_argument_list( "" );
    }
;

actual_argument_alternate_return:

```

```

    '*' INTEGER
;

return_statement:
    RW_RETURN optional_expression
;

unconditional_go_to_statement:
    RW_GO_TO INTEGER
;

computed_go_to_statement:
    RW_GO_TO '(' integer_list ')' optional_comma expression
;

assigned_go_to_statement:
    RW_GO_TO IDENTIFIER
    |
    RW_GO_TO IDENTIFIER optional_comma '(' integer_list ')'
;

integer_list:
    INTEGER
    |
    integer_list ',' INTEGER
;

optional_comma:
    /* NULL */
    |
    ','
;

pause_statement:
    RW_PAUSE optional_expression
;

stop_statement:
    RW_STOP optional_expression
;

io_statement:
    open_statement
    |
    close_statement
    |
    inquire_statement
    |
    read_statement
    |
    write_statement
    |
    print_statement
    |
    backspace_statement
    |
    rewind_statement
    |
    endfile_statement
;

open_statement:
    RW_OPEN '(' control_information_list ')'
;

close_statement:
    RW_CLOSE '(' control_information_list ')'
;

```

```

inquire_statement:
    RW_INQUIRE '(' control_information_list ')'
    ;

read_statement:
    RW_READ '(' control_information_list ')' optional_io_list
    |
    RW_READ control
    |
    RW_READ control ',' io_list
    ;

write_statement:
    RW_WRITE '(' control_information_list ')' optional_io_list
    ;

print_statement:
    RW_PRINT control
    |
    RW_PRINT control ',' io_list
    ;

backspace_statement:
    RW_BACKSPACE '(' control_information_list ')'
    |
    RW_BACKSPACE control
    ;

rewind_statement:
    RW_REWIND '(' control_information_list ')'
    |
    RW_REWIND control
    ;

endfile_statement:
    RW_ENDFILE '(' control_information_list ')'
    |
    RW_ENDFILE control
    ;

control_information_list:
    control_information
    |
    control_information_list ',' control_information
    ;

control_information:
    control
    |
    IDENTIFIER '=' expression
    ;

control:
    variable
    |
    constant
    |
    '*'
    ;

optional_io_list:
    /* NULL */
    |
    io_list
    ;

io_list:
    io
    |
    io_list ',' io

```

```

;

io:
    expression
    |
    io_implied_do_list
;

io_implied_do_list:
    '(' io_list ',' io_identifier '=' expression_list ')'

io_identifier:
    IDENTIFIER
    {
        if ( enable )
            usage_identifier( $1, SET );
    }
;

format_statement:
    RW_FORMAT
;

%%

FILE: usage/usage/include/list.h

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#define LIST struct list
LIST
{
    char *identifier;
    int line;
    int usage;
    LIST *actual_argument_list;
    LIST *next;
};

extern LIST *end_list( );
extern void add_end_list( );
extern LIST *find_list( );
extern void add_formal_argument_list( );
extern void add_call_list( );
extern void add_actual_argument_list( );
extern void add_variable_list( );
extern void usage_identifier( );
extern void begin_block( );
extern void end_block( );
extern void usage_actual_argument_list( );
extern void usage_formal_argument_list( );
extern int find_block_number( );
extern void usage_block( );

#define MAXIMUM_NUMBER_BLOCK 1024
extern char *block[ MAXIMUM_NUMBER_BLOCK ];
extern LIST *formal_argument_list[ MAXIMUM_NUMBER_BLOCK ];
extern LIST *variable_list[ MAXIMUM_NUMBER_BLOCK ];
extern LIST *call_list[ MAXIMUM_NUMBER_BLOCK ];
extern int number_block;

FILE: usage/usage/include/usage.h

/*

```

```

* Copyright 1991
* Georgia Institute of Technology
* Computer Engineering Research Laboratory
* Author: Stephen R. Wachtel
*/

#define REF 0x10000000
#define SET 0x20000000

FILE: usage/usage/library/Makefile

#
# Copyright 1991
# Georgia Institute of Technology
# Computer Engineering Research Laboratory
# Author: Stephen R. Wachtel
#

CC = cc -g
INCLUDE = ../include
CFLAGS = -I$(INCLUDE)
LIBRARY = library.a

OBJECTS = \
duplicate.o \
hollerith.o \
intrinsic.o \
link_list.o \
main.o \
non_blank.o \
summary.o \
uppercase.o \
yyerror.o \
yygetc.o \
yywrap.o

$(LIBRARY): $(OBJECTS)
ar crv $(LIBRARY) $(OBJECTS)
ranlib $(LIBRARY)

.SUFFIXES: .c .o
.c.o:
$(CC) -c $(CFLAGS) $<

clean:
rm -f $(LIBRARY) $(OBJECTS)

FILE: usage/usage/library/duplicate.c

/*
* Copyright 1991
* Georgia Institute of Technology
* Computer Engineering Research Laboratory
* Author: Stephen R. Wachtel
*/

#include <stdio.h>
#include <string.h>
#include <malloc.h>

char *duplicate( string )
register char *string;
{
    register char *temporary = (char *)NULL;

    if ( string != (char *)NULL )
    {
        if ( ( temporary = (char *)malloc( strlen( string ) + 1 ) ) != (char *)NULL )
            strcpy( temporary, string );
    }
}

```

```

        else
            fprintf( stderr, "ERROR: duplicate( %s )\n", string );
    }

    return( temporary );
} /* duplicate */

```

FILE: usage/usage/library/hollerith.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include <stdio.h>

char *hollerith( string, delimiter )
register char *string;
register char delimiter;
{
    int hollerith_length;
    register int string_length = 0;

    sscanf( string, "%dh", &hollerith_length );

    string[ string_length++ ] = delimiter;
    while ( hollerith_length != 0 )
    {
        if ( ( string[ string_length ] = yyinput( ) ) == '\n' )
        {
            yyunput( string[ string_length ] );
            break;
        }

        string_length++;
        hollerith_length--;
    }
    string[ string_length++ ] = delimiter;

    string[ string_length ] = '\0';
    return( string );
} /* hollerith */

```

FILE: usage/usage/library/intrinsic.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```

extern char *duplicate( );
extern char *uppercase( );

```

```

static char *intrinsic_table[ ] =
{
    "ABS",
    "ACOS",
    "AIMAG",
    "AINT",
    "ALOG",
    "ALOG10",
    "AMAX0",
    "AMAX1",
    "AMIN0",
    "AMIN1",
    "AMOD",
    "ANINT",
    "ASIN",
    "ATAN",

```



```

"ATAN2",
"CABS",
"CCOS",
"CEXP",
"CHAR",
"CLOG",
"CMPLX",
"CONJG",
"COS",
"COSH",
"CSIN",
"CSQRT",
"DABS",
"DACOS",
"DASIN",
"DATAN",
"DATAN2",
"DBLE",
"DCOS",
"DCOSH",
"DDIM",
"DEXP",
"DIM",
"DINT",
"DLOG",
"DLOG10",
"DMAX1",
"DMIN1",
"DMOD",
"DNINT",
"DPROD",
"DSIGN",
"DSIN",
"DSINH",
"DSQRT",
"DTAN",
"DTANH",
"EXP",
"FLOAT",
"IABS",
"ICHAR",
"IDIM",
"IDINT",
"IDNINT",
"IFIX",
"INDEX",
"INT",
"ISIGN",
"LEN",
"LGE",
"LGT",
"LE",
"LLT",
"LOG",
"LOG10",
"MAX",
"MAX0",
"MAX1",
"MIN",
"MIN0",
"MIN1",
"MOD",
"NINT",
"REAL",
"SIGN",
"SIN",
"SINH",
"SNGL",
"SQRT",
"TAN",
"TANH"
};

#define INTRINSIC_TABLE ( sizeof( intrinsic_table ) / sizeof( char * ) )

int intrinsic( identifier )
register char *identifier;
{
    register int low, high;

```

```

register int middle, test;
register char *temporary = duplicate( identifier );

low = 0;
high = INTRINSIC_TABLE - 1;

uppercase( temporary );

while ( low <= high )
{
    middle = ( low + high ) / 2;
    test = strcmp( temporary, intrinsic_table[ middle ] );

    if ( test < 0 )
    {
        high = middle - 1;
        continue;
    }

    if ( test > 0 )
    {
        low = middle + 1;
        continue;
    }

    free( temporary );
    return( 1 );
}

free( temporary );
return( 0 );
} /* intrinsic */

```

FILE: usage/usage/library/link\_list.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include <stdio.h>
#include <malloc.h>
#include "usage.h"
#include "list.h"

extern int yylineno;

char *block[ MAXIMUM_NUMBER_BLOCK ];
LIST *formal_argument_list[ MAXIMUM_NUMBER_BLOCK ];
LIST *variable_list[ MAXIMUM_NUMBER_BLOCK ];
LIST *call_list[ MAXIMUM_NUMBER_BLOCK ];
int number_block = 0;

LIST *end_list( list )
register LIST *list;
{
    if ( list != (LIST *)NULL )
    {
        while ( list->next != (LIST *)NULL )
            list = list->next;
    }

    return( list );
} /* end_list */

void add_end_list( list, identifier )
register LIST **list;
register char *identifier;
{
    register LIST *temporary = (LIST *)malloc( sizeof( LIST ) );

    temporary->identifier = identifier;

```

```

temporary->line = yylineno;
temporary->usage = 0;
temporary->actual_argument_list = (LIST *)NULL;
temporary->next = (LIST *)NULL;

if ( *list == (LIST *)NULL )
    *list = temporary;
else
    end_list( *list )->next = temporary;
} /* add_end_list */

LIST *find_list( list, identifier )
register LIST *list;
register char *identifier;
{
    while ( list != (LIST *)NULL )
    {
        if ( strcmp( list->identifier, identifier ) == 0 )
            return( list );

        list = list->next;
    }

    return( (LIST *)NULL );
} /* find_list */

void add_formal_argument_list( identifier )
register char *identifier;
{
    add_end_list( &formal_argument_list[ number_block ], identifier );
} /* add_formal_argument_list */

void add_call_list( identifier )
register char *identifier;
{
    add_end_list( &call_list[ number_block ], identifier );
} /* add_call_list */

void add_actual_argument_list( identifier )
register char *identifier;
{
    add_end_list( &end_list( call_list[ number_block ] )->actual_argument_list, identifier );
} /* add_actual_argument_list */

void add_variable_list( identifier )
register char *identifier;
{
    if ( find_list( variable_list[ number_block ], identifier ) == (LIST *)NULL )
        add_end_list( &variable_list[ number_block ], identifier );
} /* add_variable_list */

void usage_identifier( identifier, usage )
register char *identifier;
register int usage;
{
    register LIST *temporary;

    temporary = find_list( formal_argument_list[ number_block ], identifier );
    if ( temporary != (LIST *)NULL )
    {
        temporary->usage |= usage;
        return;
    }

    add_variable_list( identifier );

    temporary = find_list( variable_list[ number_block ], identifier );
    if ( temporary != (LIST *)NULL )
    {
        temporary->usage |= usage;
        return;
    }
} /* usage_identifier */

```

```

void begin_block( identifier )
register char *identifier;
{
    block[ number_block ] = identifier;
    formal_argument_list[ number_block ] = (LIST *)NULL;
    variable_list[ number_block ] = (LIST *)NULL;
    call_list[ number_block ] = (LIST *)NULL;
} /* begin_block */

void end_block( )
{
    number_block++;
} /* end_block */

void usage_actual_argument_list( block_number, actual_argument_list )
register int block_number;
register LIST *actual_argument_list;
{
    register LIST *temporary;

    while ( actual_argument_list != (LIST *)NULL )
    {
        if ( strcmp( actual_argument_list->identifier, "" ) != 0 )
        {
            temporary = find_list( formal_argument_list[ block_number ],
actual_argument_list->identifier );
            if ( temporary != (LIST *)NULL )
            {
                temporary->usage |= actual_argument_list->usage;
                actual_argument_list = actual_argument_list->next;
                continue;
            }

            temporary = find_list( variable_list[ block_number ], actual_argument_list-
>identifier );
            if ( temporary != (LIST *)NULL )
            {
                temporary->usage |= actual_argument_list->usage;
                actual_argument_list = actual_argument_list->next;
                continue;
            }
        }

        actual_argument_list = actual_argument_list->next;
    }
} /* usage_actual_argument_list */

void usage_formal_argument_list( actual_argument_list, formal_argument_list )
register LIST *actual_argument_list;
register LIST *formal_argument_list;
{
    while ( ( actual_argument_list != (LIST *)NULL ) && ( formal_argument_list != (LIST
*)NULL ) )
    {
        if ( strcmp( actual_argument_list->identifier, "" ) != 0 )
            actual_argument_list->usage |= formal_argument_list->usage;

        actual_argument_list = actual_argument_list->next;
        formal_argument_list = formal_argument_list->next;
    }
} /* usage_formal_argument_list */

int find_block_number( identifier )
register char *identifier;
{
    register int block_number;

    for ( block_number = 0; block_number != number_block; block_number++ )
    {
        if ( strcmp( block[ block_number ], identifier ) == 0 )
            return( block_number );
    }

    fprintf( stderr, "WARNING: block %s not found\n", identifier );
    begin_block( identifier );
    end_block( );
}

```

```

    return( number_block - 1 );
} /* find_block_number */

void usage_block( block_number, actual_argument_list )
register int block_number,
register LIST *actual_argument_list;
{
    register LIST *call = call_list[ block_number ];

    while ( call != (LIST *)NULL )
    {
        usage_block( find_block_number( call->identifier ), call->actual_argument_list );
        usage_actual_argument_list( block_number, call->actual_argument_list );

        call = call->next;
    }

    usage_formal_argument_list( actual_argument_list, formal_argument_list[ block_number ] );
} /* usage_block */

```

FILE: usage/usage/library/main.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include <stdio.h>

extern FILE *yyin;
extern FILE *yyout;

#define PROGRAM_argument[ 0 ]
#define INPUT_FILE_argument[ 1 ]
#define OUTPUT_FILE_argument[ 2 ]

int main( number_argument, argument )
int number_argument;
char *argument[ ];
{
    if ( number_argument == 1 )
    {
        yyin = stdin;
        yyout = stdout;

        yyparse();
        exit( 0 );
    }

    if ( number_argument == 3 )
    {
        if ( ( yyin = fopen( INPUT_FILE, "r" ) ) == (FILE *)NULL )
        {
            fprintf( stderr, "%s: ERROR - unable to open input file '%s'\n", PROGRAM,
INPUT_FILE );
            exit( -1 );
        }

        if ( ( yyout = fopen( OUTPUT_FILE, "w" ) ) == (FILE *)NULL )
        {
            fprintf( stderr, "%s: ERROR - unable to open output file '%s'\n", PROGRAM,
OUTPUT_FILE );
            exit( -1 );
        }

        yyparse();
        exit( 0 );
    }

    fprintf( stderr, "usage: %s <input file> <output file>\n", PROGRAM );
}

```

```
    exit( 0 );
} /* main */
```

FILE: usage/usage/library/non\_blank.c

```
/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include <string.h>

char *non_blank( string )
register char *string;
{
    register int offset;
    register int length;

    length = strlen( string ) - 1;
    while ( ( string[ length ] == ' ' ) && ( string[ length ] != '\0' ) )
        string[ length-- ] = '\0';

    offset = 0;
    while ( ( string[ offset ] == ' ' ) && ( string[ offset ] != '\0' ) )
        string[ offset++ ] = '\0';

    strcpy( string, &string[ offset ] );

    if ( strlen( string ) != 0 )
        return( string );
    else
        return( 0 );
} /* non_blank */
```

FILE: usage/usage/library/summary.c

```
/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include <stdio.h>
#include "usage.h"
#include "list.h"

extern FILE *yyin;
extern FILE *yyout;

void print_list( file, string, list )
register FILE *file;
register char *string;
register LIST *list;
{
    int column = strlen( string );

    fprintf( file, "%s", string );

    while ( list != (LIST *)NULL )
    {
        fprintf( file, "%s", list->identifier );
        column += strlen( list->identifier );

        if ( list->usage != 0 )
        {
            fprintf( file, "(" );
            column++;

            if ( ( list->usage & SET ) == SET )
```

```
if ( ( list->usage & REF ) == REF )
```

524 Annual Report: Digital Emulation Technology Laboratory Volume 1, Part 2

```

    {
        fprintf( file, "S" );
        column++;
    }

    if ( ( list->usage & REF ) == REF )
    {
        fprintf( file, "R" );
        column++;
    }

    fprintf( file, ")" );
    column++;
}

if ( list->next != (LIST *)NULL )
{
    fprintf( file, "," );
    column++;
}

#define MAXIMUM_COLUMN 79
    if ( column >= MAXIMUM_COLUMN )
    {
        fprintf( file, "\n" );
        fprintf( file, "\t\t" );
        column = 8;
    }

    list = list->next;
}
} /* print_list */

void print_call_list( file, list, usage )
register FILE *file;
register LIST *list;
register int usage;
{
    register char string[ 256 ];

    while ( list != (LIST *)NULL )
    {
        if ( ( usage == 0 ) || ( usage == list->usage ) )
        {
            sprintf( string, "    line %d, call %s(", list->line, list->identifier );
            print_list( file, string, list->actual_argument_list );
            fprintf( file, ")\n" );
        }

        list = list->next;
    }
} /* print_call_list */

void print_variable_list( file, block_number )
register FILE *file;
register int block_number;
{
    register LIST *list = variable_list[ block_number ];

    while ( list != (LIST *)NULL )
    {
        if ( ( list->usage & SET ) == SET )
            fprintf( file, "%s S\n", list->identifier );
        else
        {
            fprintf( file, "%s R\n", list->identifier );
        }

        list = list->next;
    }
} /* print_variable_list */

void summary( )
{
    char string[ 256 ];
    register int block_number;

```

```

usage_block( 0, 0 );

#ifdef DEBUG
for ( block_number = 0; block_number != number_block; block_number++ )
{
    sprintf( string, "%s(", block[ block_number ] );
    print_list( yyout, string, formal_argument_list[ block_number ] );
    fprintf( yyout, ")\n" );

    print_call_list( yyout, call_list[ block_number ], 0 );
    print_variable_list( yyout, block_number );
    fprintf( yyout, "\n" );
}
#else
print_variable_list( yyout, 0 );
#endif
} /* summary */

```

FILE: usage/usage/library/uppercase.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

char *uppercase( string )
register char *string;
{
    register int index = 0;

    while ( string[ index ] != '\0' )
    {
        string[ index ] = toupper( string[ index ] );
        index++;
    }

    return( string );
} /* uppercase */

```

FILE: usage/usage/library/yyerror.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

#include <stdio.h>

extern int yylineno;

void yyerror( string )
register char *string;
{
    fprintf( stderr, "line %d, %s\n", yylineno, string );

    exit( -1 );
} /* yyerror */

```

FILE: usage/usage/library/yygetc.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```



```

#include <stdio.h>
#include <ctype.h>

extern int yylineno;

int tab( length )
register int length;
{
    while ( length-- != 0 )
        yyunput( ' ' );

    return( ' ' );
} /* tab */

int yygetc( file )
register FILE *file;
{
    int c;
    int column[ 6 ];

loop:
    if ( ( c = getc( file ) ) == '\t' )
        c = tab( 6 );

    if ( c != '\n' )
        return( c );

    if ( ( column[ 0 ] = getc( file ) ) != ' ' )
        goto abort_0;
    if ( ( column[ 1 ] = getc( file ) ) != ' ' )
        goto abort_1;
    if ( ( column[ 2 ] = getc( file ) ) != ' ' )
        goto abort_2;
    if ( ( column[ 3 ] = getc( file ) ) != ' ' )
        goto abort_3;
    if ( ( column[ 4 ] = getc( file ) ) != ' ' )
        goto abort_4;
    if ( !space( column[ 5 ] = getc( file ) ) )
        goto abort_5;

    yylineno++;
    goto loop;

abort_5:
    if ( column[ 5 ] == '\t' )
        tab( 1 );
    else
    {
        yyunput( column[ 5 ] );
        if ( column[ 5 ] == '\n' )
            yylineno++;
    }

abort_4:
    if ( column[ 4 ] == '\t' )
        tab( 2 );
    else
    {
        yyunput( column[ 4 ] );
        if ( column[ 4 ] == '\n' )
            yylineno++;
    }

abort_3:
    if ( column[ 3 ] == '\t' )
        tab( 3 );
    else
    {
        yyunput( column[ 3 ] );
        if ( column[ 3 ] == '\n' )
            yylineno++;
    }

abort_2:
    if ( column[ 2 ] == '\t' )
        tab( 4 );
    else

```

```

    {
        yyunput( column[ 2 ] );
        if ( column[ 2 ] == '\n' )
            yylineno++;
    }

abort_1:
    if ( column[ 1 ] == '\t' )
        tab( 5 );
    else
    {
        yyunput( column[ 1 ] );
        if ( column[ 1 ] == '\n' )
            yylineno++;
    }

abort_0:
    if ( column[ 0 ] == '\t' )
        tab( 6 );
    else
    {
        yyunput( column[ 0 ] );
        if ( column[ 0 ] == '\n' )
            yylineno++;
    }

    return( c );
} /* yygetc */

```

FILE: usage/usage/library/yywrap.c

```

/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */

```

```

int yywrap( )
{
    return( 1 );
} /* yywrap */

```

FILE: usage/usage/scanner.l

```

%{
/*
 * Copyright 1991
 * Georgia Institute of Technology
 * Computer Engineering Research Laboratory
 * Author: Stephen R. Wachtel
 */
%}

```

```

%a 10000
%e 10000
%k 10000
%n 10000
%o 10000
%p 10000

```

```

a  [aA]
b  [bB]
c  [cC]
d  [dD]
e  [eE]
f  [fF]
g  [gG]
h  [hH]
i  [iI]
j  [jJ]
k  [kK]
l  [lL]
m  [mM]

```

```

n  [nN]
o  [oO]
p  [pP]
q  [qQ]
r  [rR]
s  [sS]
t  [tT]
u  [uU]
v  [vV]
w  [wW]
x  [xX]
y  [yY]
z  [zZ]

```

```

%{
#include "grammar.h"
extern char *yyval;

```

```

#undef YYLMAX
#define YYLMAX (256*20)

```

```

extern char *duplicate( );
extern char *hollerith( );
extern char *non_blank( );
extern char *uppercase( );
%!

```

```

%%

```

```

^[\\*cC].*[\\n]  |
^[\\ ]*[\\n]  {
#ifdef DEBUG
    ECHO;
#endif
    yyval = duplicate( yytext );
    return( COMMENT );
}

```

```

[\\ ]  {
#ifdef DEBUG
    ECHO;
#endif
    /* return( '\\ ' ) */;
}

```

```

[\\&]  {
#ifdef DEBUG
    ECHO;
#endif
    return( '\\&' );
}

```

```

[\\(]  {
#ifdef DEBUG
    ECHO;
#endif
    return( '\\(' );
}

```

```

[\\)]  {
#ifdef DEBUG
    ECHO;
#endif
    return( '\\)' );
}

```

```

[\\*]  {
#ifdef DEBUG
    ECHO;
#endif
    return( '\\*' );
}

```

```

    }

    [\*][\*] {
#ifdef DEBUG
        ECHO;
#endif
        return( EXPONENTIATE );
    }

```

```

    [\+] {
#ifdef DEBUG
        ECHO;
#endif
        return( '\+' );
    }

```

```

    [\,] {
#ifdef DEBUG
        ECHO;
#endif
        return( '\,' );
    }

```

```

    [\-] {
#ifdef DEBUG
        ECHO;
#endif
        return( '\-' );
    }

```

```

    [\.] {
#ifdef DEBUG
        ECHO;
#endif
        return( '\.' );
    }

```

```

    [\/] {
#ifdef DEBUG
        ECHO;
#endif
        return( '\/' );
    }

```

```

    [\:] {
#ifdef DEBUG
        ECHO;
#endif
        return( '\:' );
    }

```

```

    [\=] {
#ifdef DEBUG
        ECHO;
#endif
        return( '\=' );
    }

```

```

    [\n] {
#ifdef DEBUG
        ECHO;
#endif
        /* return( '\n' ) */;
    }

```

```

    [\t] {
#ifdef DEBUG
        ECHO;
#endif
        /* return( '\t' ) */;
    }

```

```
[\.]{a}{n}{d}[\.] {  
#ifdef DEBUG  
    ECHO;  
#endif  
    return( RW_AND );  
}
```

```
[\.]{e}{q}[\.] {  
#ifdef DEBUG  
    ECHO;  
#endif  
    return( RW_EQ );  
}
```

```
[\.]{e}{q}{v}[\.] {  
#ifdef DEBUG  
    ECHO;  
#endif  
    return( RW_EQV );  
}
```

```
[\.]{f}{a}{l}{s}{e}[\.] {  
#ifdef DEBUG  
    ECHO;  
#endif  
    return( RW_FALSE );  
}
```

```
[\.]{g}{e}[\.] {  
#ifdef DEBUG  
    ECHO;  
#endif  
    return( RW_GE );  
}
```

```
[\.]{g}{t}[\.] {  
#ifdef DEBUG  
    ECHO;  
#endif  
    return( RW_GT );  
}
```

```
[\.]{l}{e}[\.] {  
#ifdef DEBUG  
    ECHO;  
#endif  
    return( RW_LE );  
}
```

```
[\.]{l}{t}[\.] {  
#ifdef DEBUG  
    ECHO;  
#endif  
    return( RW_LT );  
}
```

```
[\.]{n}{e}[\.] {  
#ifdef DEBUG  
    ECHO;  
#endif  
    return( RW_NE );  
}
```

```
[\.]{n}{e}{q}{v}[\.] {  
#ifdef DEBUG  
    ECHO;  
#endif  
    return( RW_NEQV );  
}
```

```

[\\.] {n} {o} {t} [\\.] {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_NOT );
}

[\\.] {o} {r} [\\.] {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_OR );
}

[\\.] {t} {r} {u} {e} [\\.] {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_TRUE );
}

{a} {s} {s} {i} {g} {n} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_ASSIGN );
}

{b} {a} {c} {k} {s} {p} {a} {c} {e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_BACKSPACE );
}

{b} {l} {o} {c} {k} [\\ ] * {d} {a} {t} {a} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_BLOCK_DATA );
}

{c} {a} {l} {l} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_CALL );
}

{c} {h} {a} {r} {a} {c} {t} {e} {r} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_CHARACTER );
}

{c} {l} {o} {s} {e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_CLOSE );
}

{c} {o} {m} {m} {o} {n} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_COMMON );
}

```

```

(c){o}{m}{p}{l}{e}{x} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_COMPLEX );
}

(c){o}{n}{t}{i}{n}{u}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_CONTINUE );
}

(d){a}{t}{a} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_DATA );
}

(d){i}{m}{e}{n}{s}{i}{o}{n} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_DIMENSION );
}

(d){o} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_DO );
}

(d){o}{u}{b}{l}{e}{\ }*(p){r}{e}{c}{i}{s}{i}{o}{n} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_DOUBLE_PRECISION );
}

(e){l}{s}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_ELSE );
}

(e){l}{s}{e}{\ }*(i){f} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_ELSE_IF );
}

(e){n}{d} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_END );
}

(e){n}{d}{\ }*(i){f} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_END_IF );
}

(e){n}{d}{f}{i}{l}{e} {

```

```

#ifdef DEBUG
    ECHO;
#endif
    return( RW_ENDFILE );
}

{e}{n}{t}{r}{y} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_ENTRY );
}

{e}{q}{u}{i}{v}{a}{l}{l}{e}{n}{c}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_EQUIVALENCE );
}

{e}{x}{t}{e}{r}{n}{a}{l} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_EXTERNAL );
}

{f}{o}{r}{m}{a}{t}.* {
#ifdef DEBUG
    ECHO;
#endif
    yyval = duplicate( yytext );
    return( RW_FORMAT );
}

{f}{u}{n}{c}{t}{i}{o}{n} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_FUNCTION );
}

{g}{o}{[ \ ]*(t){o} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_GO_TO );
}

{i}{f} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_IF );
}

{i}{m}{p}{l}{i}{c}{i}{t} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_IMPLICIT );
}

{i}{n}{c}{l}{u}{d}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_INCLUDE );
}

{i}{n}{q}{u}{i}{r}{e} {

```



```

#ifdef DEBUG
    ECHO;
#endif
    return( RW_INQUIRE );
}

{i}{n}{t}{e}{g}{e}{r} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_INTEGER );
}

{i}{n}{t}{r}{i}{n}{s}{i}{c} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_INTRINSIC );
}

{l}{o}{g}{i}{c}{a}{l} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_LOGICAL );
}

{n}{a}{m}{e}{l}{i}{s}{t} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_NAMELIST );
}

{o}{p}{e}{n} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_OPEN );
}

{p}{a}{r}{a}{m}{e}{t}{e}{r} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_PARAMETER );
}

{p}{a}{u}{s}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_PAUSE );
}

{p}{r}{i}{n}{t} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_PRINT );
}

{p}{r}{o}{g}{r}{a}{m} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_PROGRAM );
}

{r}{e}{a}{d} {
#ifdef DEBUG

```

```

    ECHO;
#endif
    return( RW_READ );
}

{r}{e}{a}{l} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_REAL );
}

{r}{e}{t}{u}{r}{n} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_RETURN );
}

{r}{e}{w}{i}{n}{d} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_REWIND );
}

{s}{a}{v}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_SAVE );
}

{s}{t}{o}{p} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_STOP );
}

{s}{u}{b}{r}{o}{u}{t}{i}{n}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_SUBROUTINE );
}

{t}{h}{e}{n} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_THEN );
}

{t}{o} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_TO );
}

{w}{r}{i}{t}{e} {
#ifdef DEBUG
    ECHO;
#endif
    return( RW_WRITE );
}

{u}{n}{d}{e}{f}{i}{n}{e}{d} {
#ifdef DEBUG
    ECHO;

```

```

#endif
    return( RW_UNDEFINED );
}

[%a-zA-Z][_a-zA-Z0-9]* {
#ifdef DEBUG
    ECHO;
#endif
    yylval = duplicate( uppercase( yytext ) );
    return( IDENTIFIER );
}

^[0-9 ][0-9 ][0-9 ][0-9 ][0-9 ][\ ] {
#ifdef DEBUG
    ECHO;
#endif
    yylval = duplicate( non_blank( yytext ) );
    return( LABEL );
}

[0-9]+ |
[0-9]+/\.[a-zA-Z]+\.\. {
#ifdef DEBUG
    ECHO;
#endif
    yylval = duplicate( yytext );
    return( INTEGER );
}

[0-9]+\.[0-9]*([eE][\+\-]?[0-9]+)? |
[0-9]*\.[0-9]+([eE][\+\-]?[0-9]+)? |
[0-9]+([eE][\+\-]?[0-9]+)? {
#ifdef DEBUG
    ECHO;
#endif
    yylval = duplicate( yytext );
    return( REAL );
}

[0-9]+\.[0-9]*([dD][\+\-]?[0-9]+)? |
[0-9]*\.[0-9]+([dD][\+\-]?[0-9]+)? |
[0-9]+([dD][\+\-]?[0-9]+)? {
#ifdef DEBUG
    ECHO;
#endif
    yylval = duplicate( yytext );
    return( DOUBLE_PRECISION );
}

\[^\']*\\' |
\[^\"]*\\\" {
#ifdef DEBUG
    ECHO;
#endif
    yytext[ 0 ] = '\\';
    yytext[ strlen( yytext ) - 1 ] = '\\';
    yylval = duplicate( yytext );
    return( STRING );
}

[0-9]+[hH] {
#ifdef DEBUG
    ECHO;
#endif
    yylval = duplicate( hollerith( yytext, '\\' ) );
    return( HOLLERITH );
}

```